**Journal of Scheduling manuscript No.**
(will be inserted by the editor)

# Exact algorithms for single-machine scheduling with time windows and precedence constraints

**Morteza Davari · Erik Demeulemeester**
**Roel Leus · Fabrice Talla Nobibon**

Received: date / Accepted: date

**Abstract** We study a single-machine scheduling problem that is a generalization of a number of problems for which computational procedures have already been published. Each job has a processing time, a release date, a due date, a deadline and a weight representing the penalty per unit-time delay beyond the due date. The goal is to schedule all jobs such that the total weighted tardiness penalty is minimized and both the precedence constraints as well as the time windows (implied by the release dates and the deadlines) are respected. We develop a branch-and-bound algorithm that solves the problem to optimality. Computational results show that our approach is effective in solving medium-sized instances, and that it compares favorably with existing methods for special cases of the problem.

**Keywords** single-machine scheduling · branch and bound · mixed-integer programming

M. Davari ✉
Research Center for Operations Management, KU Leuven, Belgium
tel.: +32 16 37 63 43
fax: +32 16 32 66 24
E-mail: morteza.davari@kuleuven.be

E. Demeulemeester
Research Center for Operations Management, KU Leuven
E-mail: erik.demeulemeester@kuleuven.be

R. Leus
ORSTAT, KU Leuven
E-mail: roel.leus@kuleuven.be

F. Talla Nobibon
FedEx Express Europe, Middle East, Indian Subcontinent & Africa
E-mail: ftallanobibon@fedex.com

## 1 Introduction

Scheduling problems arise in production planning [40], in balancing processes [39], in telecommunication [30] and more generally in all situations in which scarce resources are to be allocated to jobs over time [34]. Depending on the application, the corresponding scheduling problem can be such that each job must be processed within a given time window, where the lower bound (*release date* or *ready time*) of this time window represents the earliest start of the execution of the job and the upper bound (*deadline*) corresponds with the latest acceptable completion time, for instance the ultimate delivery time agreed upon with the customer [16, 33, 49]. For some of these applications, only release dates or only deadlines are considered [22, 32, 35, 42]. In practice, a job often also needs to be processed before or after other jobs, e.g., due to tool or fixture restrictions or for other case-dependent technological reasons, which leads to *precedence constraints* [27, 36, 44]. Finally, the contract with a client can also contain clauses that stipulate that penalties must be paid when the execution of a job is not completed before a reference date (*due date*) [1, 13, 21, 22, 41, 42].

In this article, we develop exact algorithms for a single-machine scheduling problem with total weighted tardiness (TWT) penalties. In the standard three-field notation introduced by Graham et al. [17], the problem that we tackle can be denoted as $1|r_j, \delta_j, prec|\sum w_j T_j$: the execution of each job is constrained to take place within a time window, and we assume the corresponding deadline to be greater than or equal to a due date, which is the reference for computing the tardiness of the job. The scheduling decisions are also subject to precedence constraints. In the following lines we briefly summarize the state of the art.

Abdul-Razaq et al. [2] survey different branch-and-bound (B&B) algorithms for $1||\sum w_jT_j$. A benchmark algorithm is the B&B procedure of Potts and Van Wassenhove [38]; an older reference is Held and Karp [19], who present a dynamic programming (DP) approach. Abdul-Razaq and Potts [1] introduce a DP-based approach to obtain tight lower bounds for the generalized version of the problem where the cost function is piecewise linear. They examine their lower bounds in a B&B algorithm and solve small instances (with at most 25 jobs) to optimality. Ibaraki and Nakamura [21] extend their work and construct an exact method, called Successive Sublimation Dynamic Programming (SSDP), which solves medium-sized instances (with up to 50 jobs). Tanaka et al. [43] improve the SSDP of [21] and succeed in solving reasonably large instances (with up to 300 jobs) of $1||\sum w_jT_j$ within acceptable runtimes.

Single-machine scheduling for TWT with (possibly unequal) release dates ($1|r_j|\sum w_jT_j$) has also been studied by several authors. Akturk and Ozdemir [3, 4] and Jouglet et al. [22] develop B&B algorithms that solve small instances. Van den Akker et al. [48] propose a time-indexed formulation and a method based on column generation to solve this problem with identical processing times. Tanaka and Fujikuma [42] present an SSDP algorithm that can solve instances of $1|r_j|\sum w_jT_j$ with up to 100 jobs.

There are only few papers dealing with single-machine scheduling with deadlines and/or precedence constraints. Among these, we cite Posner [35] and Pan [32], who propose B&B algorithms for $1|\delta_j|\sum w_jC_j$, Pan and Shi [33], who develop a B&B algorithm to solve $1|r_j,\delta_j|\sum w_jC_j$, Lawler [27] and Potts [36], who present B&B algorithms to solve $1|prec|\sum w_jC_j$, and Tang et al. [45], who propose a hybrid backward and forward dynamic-programming-based Lagrangian relaxation to compute upper and lower bounds for $1|prec|\sum w_jT_j$. Tanaka and Sato [44] also propose an SSDP algorithm to solve a generalization of $1|prec|\sum w_jT_j$ (piecewise linear cost function). To the best of our knowledge, scheduling problems with release dates, deadlines and precedence constraints have not yet been studied in the literature. The goal of this paper is to fill this gap and to propose efficient B&B algorithms that solve all the foregoing subproblems within limited computation times.

The remainder of this paper is structured as follows. In Section 2 we provide some definitions and a formal problem statement, while Section 3 proposes two different integer programming formulations. In Section 4 we explain the branching strategies for our B&B algorithms, while the lower bounds, the dominance rules and the initial upper bound are discussed in Section 5, 6 and 7, respectively. Computational results are reported and discussed in Section 8. We provide a summary and conclusions in Section 9.

## 2 Problem description

The jobs to be scheduled are gathered in set $N = \{1, 2, \ldots, n\}$. Job $i$ is characterized by a processing time $p_i$, a release date $r_i$, a due date $d_i$, a deadline $\delta_i$, and a weight $w_i$, which represents the cost per unit time of delay beyond $d_i$. Jobs can neither be processed before their release dates nor after their deadlines ($0 \leq r_i \leq \delta_i$). Precedence constraints are represented by a graph $G = (N', A)$, where $N' = N \cup \{0, n+1\}$, with 0 a dummy start job and $n+1$ a dummy end. Each arc $(i, j) \in A$ implies that job $i$ must be executed before job $j$ (job $i$ is a predecessor of job $j$). We will assume that $G(N', A)$ is its own transitive reduction, that is, no transitive arcs are included in $A$. Let $\mathcal{P}_i$ be the set of all predecessors of job $i$ in $A$ ($\mathcal{P}_i = \{k|(k, i) \in A\}$) and $\mathcal{Q}_j$ the set of successors of job $i$ ($\mathcal{Q}_i = \{k|(i, k) \in A\}$). We also define an associated graph $\hat{G} = (N', \hat{A})$ as the transitive closure of $G$. We assume that $\mathcal{P}_0 = \mathcal{Q}_{n+1} = \emptyset$, and that all jobs are successor of 0 and predecessor of $n+1$ in $\hat{G}$ (apart from the jobs themselves).

Throughout this paper, we use the term 'sequencing' to refer to ordering the jobs (establishing a permutation) whereas 'scheduling' means that start (or end) times are determined. We denote by $\pi$ an arbitrary sequence of jobs, where $\pi_k$ represents the job at the $k^{\text{th}}$ position in that sequence. Let $\pi^{-1}(i)$ be the position of job $i$ in $\pi$; we only consider sequences for which $\pi^{-1}(i) < \pi^{-1}(j)$ for all $(i, j) \in A$. Value $C_i$ is the completion time of job $i$. Each sequence $\pi$ implies a schedule, as follows:

$$C_{\pi_i} = \begin{cases} \max\{r_{\pi_i}, C_{\pi_{i-1}}\} + p_{\pi_i} & \text{if } i > 1 \\ r_{\pi_i} + p_{\pi_i} & \text{if } i = 1 \end{cases}$$

Equivalently, the end of job $i$ according to sequence $\pi$ can also be written as $C_i(\pi)$. We denote by $\mathcal{D}$ the set of all feasible permutations, where a permutation $\pi$ is feasible ($\pi \in \mathcal{D}$) if and only if it generates a feasible schedule, which means that

$$r_{\pi_i} + p_{\pi_i} \leq C_{\pi_i} \leq \delta_{\pi_i} \qquad \forall i \in N$$

Note that the set $\mathcal{D}$ may be empty.

The weighted tardiness associated with the job at the $i^{\text{th}}$ position in the sequence $\pi$ is given by $W(\pi_i) = w_{\pi_i}\left(C_{\pi_i} - d_{\pi_i}\right)^+$, where $x^+ = \max\{0, x\}$. A conceptual formulation of the problem P studied in this paper is

**Table 1** Job characteristics.

| Job $i$ | $p_i$ | $r_i$ | $d_i$ | $\delta_i$ | $w_i$ |
|---|---|---|---|---|---|
| Job 1 | 2 | 3 | 10 | 14 | 1 |
| Job 2 | 3 | 4 | 11 | 13 | 2 |
| Job 3 | 4 | 3 | 8 | 15 | 3 |
| Job 4 | 2 | 2 | 6 | 9 | 1 |

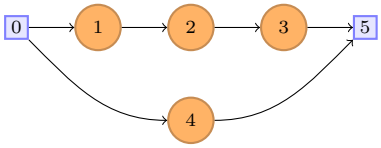


**Fig. 1:** Precedence graph $G(N', A)$.

the following:

$$\mathrm{P}: \min_{\pi \in \mathcal{D}} \mathrm{TWT}(\pi) = \sum_{i=1}^{n} W(\pi_i). \tag{1}$$

This problem is at least as hard as $1||\sum w_i T_i$, which is known to be strongly NP-hard [26,28,34]. A stronger result is that the mere verification of the existence of a feasible schedule that respects a set of ready times and deadlines is already NP-complete (problem SS1, page 236, [15]); we do not, however, incorporate the feasibility check as a formal part of the problem statement.

*Example 1* Consider the following instance of P with $n = 4$ jobs. The processing times, release dates, due dates, deadlines and weights of the jobs are given in Table 1. The graph representing the precedence constraints is depicted in Figure 1, with arc set $A = \{(0,1), (0,4), (1,2), (2,3), (3,5), (4,5)\}$.

An optimal solution to this instance is $\pi = (4,1,2,3)$, which leads to the schedule $C_1 = 6$, $C_2 = 9$, $C_3 = 13$ and $C_4 = 4$. The objective value is $w_4 \times 0 + w_1 \times 0 + w_2 \times 0 + w_3 \times (13-8) = 3 \times 5 = 15$.

## 3 Mathematical formulations

The conceptual formulation for P presented in the previous section is not linear, therefore it cannot be used by a standard (linear) mixed-integer programming (MIP) solver. In this section, we propose an Assignment Formulation (ASF) and a Time-Indexed Formulation (TIF) for the problem. These formulations are adaptations of those presented in [23,41].

### 3.1 Assignment formulation

We use binary decision variables $x_{is} \in \{0,1\}$ $(i \in N, s \in \{1,2,\ldots,n\})$, which identify the position of jobs in the sequence so that $x_{is}$ is equal to 1 if job $i$ is the $s^{\text{th}}$ job processed and equal to 0 otherwise. In other words, $x_{is} = 1$ if and only if $\pi_s = i$. We also use additional continuous variables $T_i \geq 0$ representing the tardiness of job $i \in N$ and continuous variables $\tau_s \geq 0$ representing the machine idle time immediately before the execution of the $s^{\text{th}}$ job. The MIP formulation is given by:

$$\text{ASF}: \quad \min \quad \sum_{i=1}^{n} w_i T_i \tag{2}$$

subject to

$$\sum_{s=1}^{n} x_{is} = 1 \quad \forall i \in N \tag{3}$$

$$\sum_{i=1}^{n} x_{is} = 1 \quad \forall s \in \{1,2,\ldots,n\} \tag{4}$$

$$\sum_{s=1}^{n} x_{is} s \leq \sum_{t=1}^{n} x_{jt} t - 1 \quad \forall (i,j) \in A \tag{5}$$

$$\tau_s \geq \sum_{i=1}^{n} x_{is} r_i - \sum_{t=1}^{s-1} \left( \sum_{i=1}^{n} (x_{it} p_i) + \tau_t \right) \quad \forall s \in N \tag{6}$$

$$\sum_{t=1}^{s} \tau_t + \sum_{t=1}^{s-1} \sum_{i=1}^{n} p_i x_{it} + \sum_{i=1}^{n} ((p_i - \delta_i) x_{is}) \leq 0 \quad \forall s \in N \tag{7}$$

$$T_i \geq \sum_{t=1}^{s} \tau_t + \sum_{t=1}^{s-1} \sum_{j=1}^{n} p_j x_{jt} + p_i - d_i - (1 - x_{is}) M_i \quad \forall i \in N, s \in N \tag{8}$$

$$x_{is} \in \{0,1\}, \tau_s, T_i \geq 0 \quad \forall i \in N, s \in \{1,2,\ldots,n\} \tag{9}$$

The objective function (2) is a reformulation of (1). The set of constraints (3) ensures that all jobs are executed. Constraints (4) check that each position in the sequence is occupied by exactly one job. The set of constraints (5) enforces the precedence restrictions. The set of equations (6) computes the idle time of the machine between the jobs in positions $s-1$ and $s$, and ensures that each job is not started before its release date. In this set of constraints, $\sum_{t=1}^{s-1} (\sum_{i=1}^{n} (x_{it} p_i) + \tau_t)$ equals the completion time of the $(s-1)^{\text{th}}$ job. Constraints (7) ensure that each job is not completed after its deadline, where $\sum_{t=1}^{s} \tau_t + \sum_{t=1}^{s-1} \sum_{i=1}^{n} p_i x_{it}$ is the start time of the $s^{\text{th}}$ job. Constraints (8) compute the correct value of the tardiness of job $i$, with $M_i = \delta_i - d_i$ the maximum tardiness of job $i$.

A variant of ASF is obtained by replacing the set of constraints (5) by the following:

$$\sum_{s=v}^{n} x_{is} + \sum_{s=1}^{v} x_{js} \leq 1 \quad \forall (i,j) \in A, \forall v \in \{1,\ldots,n\}. \tag{10}$$

We refer to this alternative formulation as ASF′. We have the following result:

**Lemma 1** ASF′ *is stronger than* ASF.

All proofs are included in the Appendix. The number of constraints in (10) is much higher than in (5). As a result, the additional computational effort needed to process this higher number of constraints might offset the improvement of a stronger bound, and we will empirically compare the performance of the two variants in Section 8.4.

### 3.2 Time-indexed formulation

Let $T_S$ (respectively $T_E$) be a lower (respectively upper) bound on the time the execution of any job can be completed; we compute these values as $T_S = \min\{r_i + p_i | i \in N\}$ and $T_E = \max\{\delta_i | i \in N\}$. The time-indexed formulation uses binary decision variables $x_{it} \in \{0,1\}$, for $i \in N$ and $T_S \leq t \leq T_E$, where $x_{it} = 1$ if job $i$ is completed (exactly) at time $t$ and $x_{it} = 0$ otherwise. We also introduce the set of parameters $T_{it} = (t - d_i)^+$, representing the tardiness of job $i$ when it finishes at time $t$. The time-indexed formulation is given by:

$$\text{TIF}: \quad \min \sum_{i=1}^{n} \sum_{t=r_i+p_i}^{\delta_i} w_i T_{it} x_{it} \tag{11}$$

subject to

$$\sum_{i=1}^{n} \sum_{s=\max\{t, r_i+p_i\}}^{\min\{\delta_i, t+p_i-1\}} x_{is} \leq 1 \qquad \forall t, T_S \leq t \leq T_E \tag{12}$$

$$\sum_{t=r_i+p_i}^{\delta_i} x_{it} = 1 \qquad \forall i \in N \tag{13}$$

$$\sum_{s=r_i+p_i}^{\delta_i} x_{is} s \leq \sum_{t=r_j+p_j}^{\delta_j} x_{jt} t - p_j \quad \forall (i,j) \in A \tag{14}$$

$$x_{it} \in \{0,1\} \quad i \in N,\ r_i + p_i \leq t \leq \delta_i \tag{15}$$

The set of constraints (12) eliminates the parts of the solution space where the jobs overlap. The constraint set (13) ensures that all jobs are scheduled exactly once. We enforce precedence constraints in the formulation using the set of constraints (14).

Similarly as for the assignment formulation, we introduce an alternative formulation TIF′ by replacing the set of constraints (14) by the following:

$$\sum_{s=t}^{\delta_i} x_{is} + \sum_{s=r_j+p_j}^{t-p_i} x_{js} \leq 1 \tag{16}$$

$$\forall (i,j) \in A; \forall t, \max\{r_i, r_j + p_j\} + p_i \leq t \leq \min\{\delta_i, \delta_j + p_i\}$$


**Fig. 2:** The structure of a partial schedule.

**Lemma 2 (From [5, 9])** TIF′ *is stronger than* TIF.

As explained for the assignment formulation, the performance of the new formulation is not necessarily better. In fact, it can be much worse than TIF, since in a time-indexed formulation the number of additional constraints is quite large (pseudo-polynomial).

## 4 Branching strategies

In this section we discuss two different branching strategies for our B&B algorithm. The structure of the B&B search trees is as follows: each tree consists of a finite number of nodes and branches, and at each level of the tree we make a sequencing decision for one job. Each node thus corresponds with a selection $S_P \subseteq N$ containing the already scheduled jobs and a set of unscheduled jobs $U = N \backslash S_P$. Each node also has two feasible partial sequences $\sigma_B$ and $\sigma_E$ of the scheduled jobs (each $i \in S_P$ appears in exactly one of these two): $\sigma_B$ (respectively $\sigma_E$) denotes the partial sequence of jobs scheduled from the beginning (respectively end) of the scheduling horizon; see Figure 2 for an illustration. All jobs that are not scheduled, belong to the set of unscheduled jobs $U = E_B \cup E_E \cup E_N$. $E_B$ is subset of unscheduled jobs that are eligible to be scheduled immediately after the last job in $\sigma_B$, $E_E$ is the subset of unscheduled jobs that are eligible to be scheduled immediately before the first job in $\sigma_E$ and $E_N$ is the subset of unscheduled jobs that are not in $E_B \cup E_E$.

The root node represents an empty schedule ($S_P = \sigma_B = \sigma_E = \emptyset$). Each node branches into a number of child nodes, which each correspond with the scheduling of one particular job, called the *decision job*, as early as possible after the last job in $\sigma_B$ or as late as possible before the first job in $\sigma_E$. A branch is called a *forward branch* if it schedules a job after the last job in $\sigma_B$, and is called a *backward branch* if it schedules a job before the first job in $\sigma_E$. In our branching strategies, there will be either only forward branches or only backward branches emanating from each given node. We will say that a node is of type FB (respectively BB) if all its branches are forward (respectively backward) branches.

Although scheduling jobs backward (from the end of the time horizon) often improves the tightness of lower bounds [38] when release dates are equal, it probably

decreases the quality of the lower bounds in the presence of non-equal release dates; see Section 4.2 and 5.3 for a description of backward branching and of the lower bounds, respectively, and Section 8.4 for the empirical results and a discussion. Also, the efficiency of some dominance rules may decrease when we switch from forward scheduling to backward scheduling; see Section 6.4 for more details. We propose two B&B algorithms, each applying one of the branching strategies: BB1 corresponds with branching strategy 1 where only FB nodes are used and BB2 corresponds with branching strategy 2 where both FB and BB are created. The bounding and the dominance properties discussed in the following sections are the same in both B&B algorithms.

Let $C_{\max}(\sigma)$ be the completion time of the last job in the sequence $\sigma$. Throughout the branching procedure, we maintain two vectors of *updated release dates*, namely $\hat{\mathbf{r}} = (\hat{r}_1, \ldots, \hat{r}_n)$ and $\bar{\mathbf{r}} = (\bar{r}_1, \ldots, \bar{r}_n)$, defined as follows:

$$\hat{r}_j = \max\{r_j, C_{\max}(\sigma_B)\}$$

$$\bar{r}_j = \max\left\{\hat{r}_j, \max_{k \in \mathcal{P}_j}\{\bar{r}_k + p_k\}\right\}.$$

Let $st(\pi)$ denote the start time of the first job according to sequence $\pi$. In line with the two vectors of updated release dates, we also introduce two vectors of *updated deadlines*, namely $\hat{\delta} = (\hat{\delta}_1, \ldots, \hat{\delta}_n)$ and $\bar{\delta} = (\bar{\delta}_1, \ldots, \bar{\delta}_n)$, which are recursively computed as follows:

$$\hat{\delta}_j = \min\{\delta_j, st(\sigma_E)\}$$

$$\bar{\delta}_j = \min\left\{\hat{\delta}_j, \min_{k \in \mathcal{Q}_j}\{\bar{\delta}_k - p_k\}\right\}.$$

We use these updated release dates and deadlines in computing lower bounds and dominance rules. $\bar{\delta}$ and $\bar{\mathbf{r}}$ are more restrictive than $\hat{\delta}$ and $\hat{\mathbf{r}}$ in each node of the search tree ($\bar{r}_j \geq \hat{r}_j$ and $\bar{\delta}_j \leq \hat{\delta}_j$). Although being restrictive often is positive, $\hat{r}_j$ and $\hat{\delta}_j$ are occasionally preferred over $\bar{r}_j$ and $\bar{\delta}_j$, specifically in parts of computations related to the dominance rules discussed in Section 6. Further explanations of these occasions are given in Section 6. There are many cases in which $\bar{r}_j = \hat{r}_j$ (respectively $\bar{\delta}_j = \hat{\delta}_j$) and either of the updated release dates (respectively deadlines) can be used. In these cases, we use $\hat{r}_j$ (respectively $\hat{\delta}_j$) because less computations are needed.

### 4.1 Branching strategy 1

Branching strategy 1 only uses FB nodes. The search tree is explored depth-first such that among children of a node, those with larger out-degrees (number of transitive successors) of their decision jobs in $\hat{G}$ are visited first. As a tie-breaking rule, among children with equal out-degrees of their decision jobs, the node with lower index is visited first.

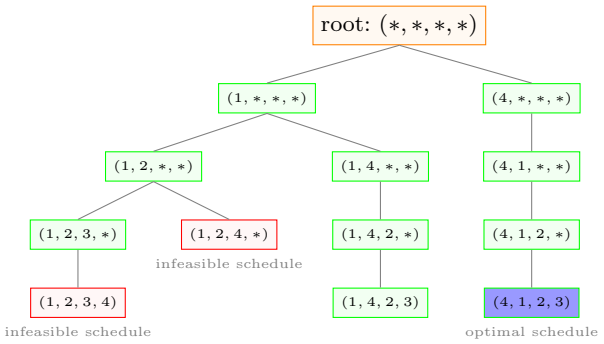


**Fig. 3:** Branching strategy 1 for Example 1 without dominance rules and without lower bounds.

Figure 3 illustrates branching strategy 1 applied to Example 1; an asterisk '*' indicates that the position has not been decided yet. Among the children of the root node, the node $(1, *, *, *)$ corresponds with the decision job (job 1) with the largest out-degree (namely 3). As a result, the node $(1, *, *, *)$ is visited first. The nodes $(2, *, *, *)$ and $(3, *, *, *)$ are not in the tree because they violate precedence constraints. Among the children of $(1, *, *, *)$, the node $(1, 2, *, *)$ is visited first because it has the decision job 2 with the largest out-degree. Among the children of $(1, 2, *, *)$, the node $(1, 2, 3, *)$ is visited first because its decision job has the largest out-degree and the smallest index. In Figure 3, green nodes are FB nodes; no BB nodes are present. Red nodes are considered infeasible because the completion of a job (namely job 4) occurs after its deadline. The node $(1, 4, 2, 3)$ corresponds with a feasible schedule, but it is not optimal: its objective value is greater than 15, which is attained by the optimal sequence $(4, 1, 2, 3)$.

### 4.2 Branching strategy 2

In branching strategy 2, we try to exploit the advantages of backward scheduling whenever possible, so the search tree consists of both FB and BB nodes. If the inequality $C_{\max}(\sigma_B) < r_{\max}(U) = \max_{j \in U}\{r_j\}$ holds, then the start times of the jobs in $\sigma_E$ will depend on the order in which unscheduled jobs are processed. Therefore, if the inequality $C_{\max}(\sigma_B) < r_{\max}(U)$ holds, the corresponding node is of type FB. Otherwise, the completion time of the last job in $\sigma_E$ can be computed regardless of the sequencing decisions for the jobs in $U$, and we have a BB node. The branching is *depth-first* for both FB and BB nodes. Among the children of an FB

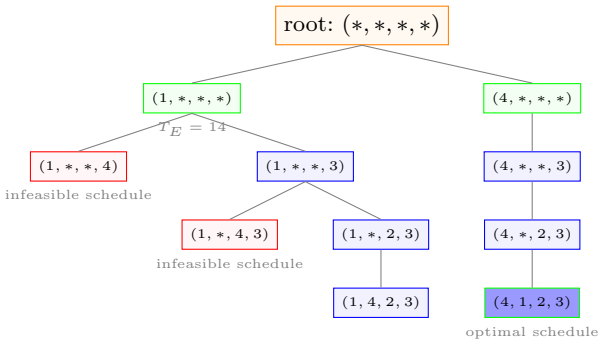


**Fig. 4:** Branching strategy 2 for Example 1 without dominance rules and without lower bounds.

(respectively BB) node, those with higher (respectively lower) out-degrees of their decision jobs are visited first. As a tie-breaking rule, among children with equal out-degrees, the node with lower (respectively higher) index is visited first.

Figure 4 illustrates branching strategy 2 for Example 1; green nodes are of type FB and blue nodes are of type BB. The root node is FB because $C_{\max}(\emptyset) = 0 < 4 = r_{\max}(\{1,2,3,4\})$. At the node labeled $(1,*,*,*)$, the completion time $C_{\max}(1,*,*,*) = 5$ of the decision job surpasses $r_{\max}(\{1,2,3,4\}) = 4$, therefore the end of scheduling horizon is computed ($T_E = 5+3+4+2 = 14$) and the node is BB. The red nodes are infeasible because the completion time of job 4 falls after its deadline.

## 5 Lower bounding

In this section we describe the lower bounds that are implemented in our B&B algorithm. Section 5.1 first introduces a conceptual formulation for our problem, Section 5.2 describes a very fast lower bounding procedure, and in Section 5.3 we describe several lower bounds based on Lagrangian relaxation.

### 5.1 Another conceptual formulation

Let variable $C_j \geq 0$ denote the completion time of job $j \in N$ and let variable $T_j \geq 0$ represent the tardiness of job $j$. An alternative formulation of our problem is given by:

$$\text{P}: \quad \min \quad \sum_{j=1}^{n} w_j T_j \tag{17}$$

subject to

$$T_j \geq C_j - d_j \qquad \forall j \in N \tag{18}$$
$$C_j \geq r_j + p_j \qquad \forall j \in N \tag{19}$$
$$C_j \leq \delta_j \qquad \forall j \in N \tag{20}$$
$$C_j \geq C_i + p_j \qquad \forall (i,j) \in A \tag{21}$$
$$C_j \geq C_i + p_j \text{ or } C_i \geq C_j + p_i \qquad \forall i,j \in N; i < j \tag{22}$$
$$T_j \geq 0 \qquad \forall j \in N \tag{23}$$
$$C_j \geq 0 \qquad \forall j \in N \tag{24}$$

In the above formulation, constraints (18) and (23) reflect the definition of job tardiness. Constraints (19) and (20) enforce time windows. Constraints (21) ensure that each job is scheduled after all its predecessors. Constraints (22) guarantee that jobs do not overlap. We will use this formulation in Section 5.3 for producing lower bounds.

To the best of our knowledge, a lower-bound procedure specifically for P has to date not been developed in the literature. Lower bounds proposed for $1||\sum w_j T_j$, $1|prec|\sum w_j C_j$ and $1|r_j|\sum w_j C_j$, however, can also function as a lower bound for P; this is shown in the following theorems. These theorems are extensions of those presented in [3].

Let $I$ be an instance of $1|\beta|\sum w_j T_j$. We construct an instance $I'$ of $1||\sum w_j T_j$ by removing all constraints implied by $\beta$ and an instance $I''$ of $1|\beta|\sum w_j C_j$ by replacing all due dates with zeros. Let $\text{TWT}^*(I)$ be the optimal objective value of $I$. Given any valid lower bound $lb_{I'}$ on the optimal value of $I'$, we have:

**Theorem 1** $lb_{I'} \leq \text{TWT}^*(I)$.

A job is called *early* if it finishes at or before its due date and is said to be *tardy* if it finishes after its due date. Let $C_j(S)$ be the completion time of job $j$ in feasible solution $S$. For an optimal solution $S^*$ to $I$, we partition $N$ into two subsets: the set $\mathcal{E}$ of early jobs and the set $\mathcal{T}$ of tardy jobs. Let $lb^E$ be a lower bound on the value $\sum_{j\in\mathcal{E}} w_j(d_j - C_j(S^*))$. Given any valid lower bound $\bar{lb}_{I''}$ on the optimal value of $I''$, we have:

**Theorem 2** $\bar{lb}_{I''} - \sum_j w_j d_j + lb^E \leq \text{TWT}^*(I)$.

In the following, we remove several combinations of constraints in P to construct subproblems for which there exist polynomial-time-bounded algorithms for computing lower bounds. These bounds then directly lead to valid lower bounds for P via Theorem 1 and Theorem 2.

### 5.2 A very fast trivial lower bound

Let $\mathrm{P_T}$ be the trivial subproblem of P in which constraints (18), (19), (20) and (21) are removed, which is then equivalent to $1||\sum w_jC_j$. An optimal solution $S^*$ to $\mathrm{P_T}$ (with the optimal value $\mathrm{OPT}(S^*)$) follows sequence $\sigma_T$, which sequences jobs according to the *shortest weighted processing time* (SWPT) rule [34]. By Theorem 1 and 2, $\mathrm{LB_T} = \mathrm{OPT}(S^*) - \sum_j w_j d_j + lb^E$ is a valid lower bound for P. We compute $lb^E$ as the summation of the earliness values when each job is scheduled at its latest possible starting time. Note that if $r_j = d_j = 0$ for all jobs $j$ and $\sigma_T$ does not violate any deadline nor precedence constraint, then $\sigma_T$ is optimal to P and $\mathrm{OPT} = \mathrm{LB_T}$. In B&B algorithms, this situation frequently occurs when some jobs have already been scheduled.

### 5.3 Lagrangian-relaxation-based bounds

In this section, we use Lagrangian relaxation for computing various lower bounds. Let $\mathrm{P_0}$ be the subproblem of P in which constraints (19), (20) and (21) are removed. This problem is studied by Potts and Van Wassenhove [38] and is considered as our base problem. Let $\lambda$ be a vector of Lagrangian multipliers. Potts and Van Wassenhove [38] obtain the following Lagrangian problem associated with $\mathrm{P_0}$:

$$\mathrm{LR_{P_0}}: \quad L_0(\lambda) = \min \sum_{j=1}^{n} (w_j - \lambda_j)T_j + \sum_{j=1}^{n} \lambda_j (C_j - d_j)$$

subject to constraints (22)–(24).

Parameter $\lambda_j$ is the Lagrangian multiplier associated with job $j$ ($0 \leq \lambda_j \leq w_j$). Potts and Van Wassenhove propose a polynomial-time algorithm to set the multipliers. Their algorithm yields a very good lower bound for $\mathrm{P_0}$; they compute the optimal values of the multipliers in $\mathrm{O}(n \log n)$ time, and for a given set of multipliers, the bound itself can be computed in linear time. Let $\lambda_{\mathrm{PV}}$ be the best Lagrangian multipliers computed by Potts and Van Wassenhove [38]; we refer to this lower bound as $\mathrm{LB_0} = L_0(\lambda_{\mathrm{PV}})$. By Theorem 1, $\mathrm{LB_0}$ is also a valid bound for P. Quite a number of aspects of the definition of P are completely ignored in $\mathrm{LB_0}$, however; in the following sections, we will examine a number of ways to strengthen $\mathrm{LB_0}$.

#### *5.3.1 Retrieving precedence constraints*

When $A \neq \emptyset$ then incorporating some or all of precedence constraints into the lower bound will improve its quality. We partition arc set $A$ as $A = A' \cup A''$,

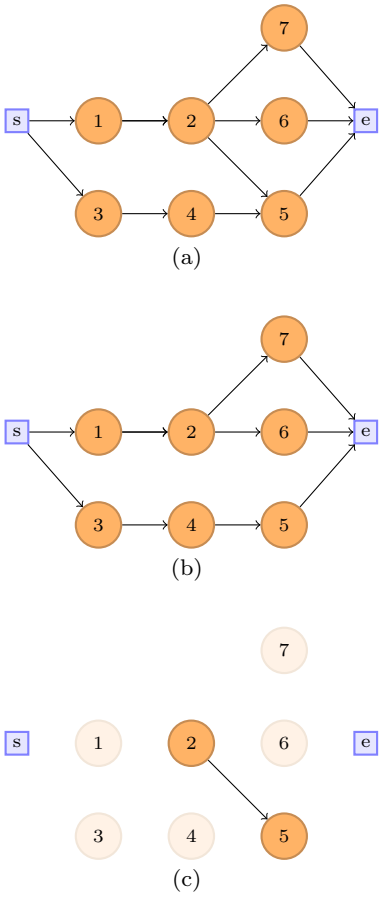


**Fig. 5:** This figure shows (a) an example graph $G$, (b) an associated VSP sub-graph $G'$ and (c) $G''$.

where $G' = (N, A')$ is a two-terminal *vertex serial-parallel* (VSP) graph and $G'' = (N, A'')$. Figure 5 depicts an example of this graph decomposition. For the precise definition of VSP graphs, we refer to Valdes et al. [46]. It should be noted that there exist two types of serial-parallel graphs: VSP graphs and *edge serial-parallel* (ESP) graphs. Valdes et al. [46] describe the link between these two types: a graph is VSP if and only if its so-called 'line-digraph inverse' is ESP.

We split the set of constraints (21) into two subsets, as follows:

$$C_j \geq C_i + p_j \qquad \forall (i,j) \in A' \tag{25}$$
$$C_j \geq C_i + p_j \qquad \forall (i,j) \in A'' \tag{26}$$

We introduce $\mathrm{P_1}$, which is a generalization of $\mathrm{P_0}$ where precedence constraints are retrieved by imposing constraints (25) and (26). We create the following associ-

ated Lagrangian problem:

$$\mathrm{LR}_{\mathrm{P}_1}: \quad L_1(\lambda,\mu) = \min \sum_{j\in N}(w_j-\lambda_j)T_j + \sum_{j\in N}\lambda_j(C_j-d_j) + \sum_{j\in N}\sum_{k\in\mathcal{Q}_j}\mu_{jk}(C_j+p_k-C_k)$$

subject to constraints (22)–(25).

Here $\lambda_j \geq 0$ is again the multiplier associated with job $j$ and $\mu_{jk} \geq 0$ denotes the Lagrangian multiplier associated with the arc $(j,k) \in A$. We deliberately keep constraints (25) in the Lagrangian problem $\mathrm{LR}_{\mathrm{P}_1}$. The objective function can be rewritten as

$$\sum_{j\in N}(w_j-\lambda_j)T_j + \sum_{j\in N} w'_j C_j + c$$

where

$$w'_j = \lambda_j + \sum_{k\in\mathcal{Q}_j}\mu_{jk} - \sum_{k\in\mathcal{P}_j}\mu_{kj}$$

and

$$c = \sum_{j\in N}\sum_{k\in\mathcal{Q}_j}\mu_{jk}p_k - \sum_{j\in N}\lambda_j d_j,$$

so it can be seen that $\mathrm{LR}_{\mathrm{P}_1}$ is a total-weighted-completion-times problem with serial-parallel precedence constraints, because all $T_j$ will be set to zero and $\sum_{j\in N}(w_j-\lambda_j)T_j$ can be removed from the formulation. Lawler [27] proposes an algorithm that solves this problem in $\mathrm{O}(n\log n)$ time provided that a *decomposition tree* is also given for the VSP graph $G'$. Valdes et al. [46] propose an $\mathrm{O}(n+m)$-time algorithm to construct a decomposition tree of a VSP graph, where $m$ is the number of arcs in the graph. Calinescu et al. [8] show that any VSP graph (directed or undirected), including $G'$, has at most $2n-3$ arcs. Therefore, for any given $\lambda$ and $\mu$, the problem $\mathrm{LR}_{\mathrm{P}_1}$ is solvable in $\mathrm{O}(n\log n)$ time. From the theory of Lagrangian relaxation (see Fisher [14]), for any choice of non-negative multipliers, $L_1(\lambda,\mu)$ provides a lower bound for $\mathrm{P}_1$. By Theorem 1, this lower bound is also valid for P. In Section 5.3.2, we explain how to choose appropriate values for $\lambda$ and $\mu$ and Section 5.3.3 describes how to select a suitable VSP graph $G'$ and how to construct a decomposition tree for $G'$.

### *5.3.2 Multiplier adjustment*

We present a two-phase adjustment (TPA) procedure for the multipliers in $L_1(\lambda,\mu)$. Let $\lambda_{\mathrm{TPA}}$ and $\mu_{\mathrm{TPA}}$ be Lagrangian multipliers adjusted by TPA; these lead to a new lower bound $\mathrm{LB}_1 = L_1(\lambda_{\mathrm{TPA}},\mu_{\mathrm{TPA}})$. The TPA procedure is heuristic, in the sense that it may not minimize $L_1$ in $\lambda$ and $\mu$.

In the first stage of TPA, we simply ignore precedence constraints altogether. For a feasible solution $S$, consider the function $g(\lambda,\mu,S)$ defined as follows:

$$g(\lambda,\mu,S) = \sum_{j\in N}(w_j-\lambda_j)T_j + \sum_{j\in N}\lambda_j(C_j-d_j) + \sum_{j\in N}\sum_{k\in\mathcal{Q}_j}\mu_{jk}(C_j+p_k-C_k).$$

We start with the Lagrangian problem $\hat{\mathrm{LR}}_{\mathrm{P}_1}$ where $\hat{L}_1(\lambda,\mu) = \min g(\lambda,\mu,S)$ subject to constraints (22)–(24), which is a relaxation of $\mathrm{LR}_{\mathrm{P}_1}$. We simply set all $\mu_{jk}$ to zero ($\mu = \mu_0 = (0,\ldots,0)$); with this choice, $\hat{L}_1(\lambda,\mu) = L_0(\lambda)$ and we set $\lambda_{\mathrm{TPA}} = \lambda_{\mathrm{PV}}$.

In the second stage of TPA, the multipliers $\mu_{jk}$ are adjusted assuming that $\lambda = \lambda_{\mathrm{TPA}}$ is predefined and constant. This adjustment is an iterative heuristic; we adopt the *quick ascent direction* (QAD) algorithm proposed by van de Velde [47]. One iteration of TPA runs in $\mathrm{O}(m+n\log n)$ time, where $m = |A|$. We have run a number of experiments to evaluate the improvement of the lower bound as a function of the number of iterations $k_{\max}$. For a representative dataset, Table 2 shows that the average percentage deviation of $\mathrm{LB}_1$ from $\mathrm{LB}_0$ significantly increases in the first iterations, whereas after about five iterations the incremental improvement becomes rather limited; more information on the choices for $k_{\max}$ follows in Sections 5.3.3 and 8.2. The instance generation scheme is explained in Section 8.1.

**Table 2** The average percentage deviation between $\mathrm{LB}_1$ and $\mathrm{LB}_0$ tested on $\mathrm{Ins}^L$.

| $k_{\max}$ | $n$ | | |
|---|---|---|---|
| | 20 | 30 | 40 |
| 0 | 11.576 | 8.505 | 6.85 |
| 5 | 14.579 | 17.454 | 13.493 |
| 10 | 15.026 | 18.147 | 14.065 |
| 20 | 15.207 | 18.419 | 14.344 |
| 50 | 15.296 | 18.503 | 14.466 |
| 100 | 15.310 | 18.508 | 14.495 |
| $\infty$ | 15.314 | 18.512 | 14.506 |

**Theorem 3** $\mathrm{LB}_0 \leq \mathrm{LB}_1$.

### *5.3.3 Finding a VSP graph*

$\mathrm{LB}_1$ requires a decomposition of graph $G$ into two subgraphs $G' = (N,A')$ and $G'' = (N,A'')$, such that $A' \cup A'' = A$ and $G'$ is a VSP graph. The more arcs we can include in $A'$, the tighter the lower bound. In

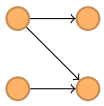

**Fig. 6:** The forbidden subgraph for VSP graphs.

the following, we discuss procedures to find a VSP subgraph $G'$ with maximum number of arcs; we refer to this problem as the *maximum VSP subgraph* (MVSP) problem.

Valdes et al. [46] state the following result:

**Lemma 3 (From [46])** *A graph $G$ is VSP if and only if its transitive closure does not contain the graph of Figure 6 as a subgraph.*

Valdes et al. refer to the pattern in Figure 6 as the *forbidden subgraph*. Polynomial-time exact procedures exist for finding an ESP subgraph with maximum number of nodes (see [6], for instance), but to the best of our knowledge, no exact approach for MVSP has been proposed yet in literature. McMahon and Lim [29] suggest a heuristic traversal procedure to find and eliminate all forbidden subgraphs and, at the same time, construct a binary decomposition tree for the resulting VSP graph. Their procedure runs in $O(n+m)$ time. The number of arcs in a VSP graph is bounded by $2n-3$ for an arbitrary non-VSP graph, but the maximum number of arcs for an arbitrary input graph is $O(n^2)$. We implement a slightly modified variant of the algorithm in [29] to compute $G'$; we select arcs for removal so that the lower bound remains reasonably tight. Simultaneously, it constructs a decomposition tree for the obtained VSP graph. The time complexity of $O(n+m)$ is maintained.

The structure of our heuristic decomposition and arc-elimination procedure is described in the following lines. The procedure constructs a decomposition tree by exploiting *parallel* and *serial node reduction* [27]. Parallel reduction merges a job pair into one single job if both jobs have the same predecessor and successor sets. In the decomposition tree, such jobs are linked by a $P$ node, which means they can be processed in parallel (see Figure 7(b)). Serial reduction merges a job pair $\{i,j\}$ into one single job if arc $(i,j) \in A$, job $i$ has only one successor and job $j$ has only one predecessor. In the decomposition tree, such two jobs are linked by an $S$ node, which means they cannot be processed in parallel (see Figure 7(d)). Whenever a forbidden subgraph is recognized, the procedure removes arcs such that the forbidden subgraph is resolved (removed) and the total number of removed arcs (including transitive and merged arcs) is approximately minimized (see Figures 7(b)–7(c)). Notice that some arcs may actually represent multiple merged arcs, so removing one arc in one iteration might imply the removal of multiple arcs simultaneously in the original network $G$.

**Table 3** The average percentage deviation between $LB_1$ and $LB_0$ tested on $Ins^L$ with 40 jobs.

| $k_{max}$ | $LB_1$(VSP) | $LB_1$(NO) |
|---|---|---|
| 0 | 6.850 | 0 |
| 1 | 10.515 | 9.057 |
| 2 | 12.171 | 11.497 |
| 3 | 12.896 | 12.538 |
| 5 | 13.493 | 13.385 |
| 10 | 14.344 | 14.020 |
| 100 | 14.466 | 14.458 |

The proposed algorithm is run only once, in the root node of the search tree. In each other node of the search tree, graphs $G'$ and $G''$ are constructed by removing from the corresponding graphs in the parent node the arcs associated with the scheduled jobs; the resulting graphs are then the input for computing $LB_1$. Notice that for each child node, both graphs $G'$ and $G''$ as well as the associated decomposition tree can be constructed in $O(n)$ time.

To evaluate the impact of our arc elimination procedure on the quality of the bounds, we examine two variations of $LB_1$, namely $LB_1(\text{VSP}) = L_1(\lambda_{\text{TPA}}, \mu_{\text{TPA}})$, where all forbidden graphs in $G$ are resolved using the arc elimination procedure, and $LB_1(\text{NO}) = \hat{L}_1(\lambda_{\text{TPA}}, \mu_{\text{TPA}})$, in which we simply remove all arcs ($A' = \emptyset$ and $A'' = A$). Let $k_{max}$ be the maximum number of iterations for TPA, as explained in Section 5.3.2. Table 3 demonstrates the success of our proposed algorithm in tightening the bound. The distance between the bounds is decreasing with increasing $k_{max}$, but in a B&B algorithm, a large value for $k_{max}$ becomes computationally prohibitive.

**Theorem 4** $LB_1(\text{NO}) \leq LB_1(\text{VSP})$ *for the same value of $k_{max}$.*

### *5.3.4 Retrieving release dates and deadlines*

Bound $LB_1$ turns out not be to be very tight when release dates are rather heterogeneous. Below, we examine two means to produce a stronger bound, namely block decomposition and job splitting.

*Block decomposition* We follow references [18,33,37] in setting up a decomposition of the job set into separate *blocks*: a *block* is a subset of jobs for which it is a dominant decision to schedule them together. We sort and renumber all jobs in non-decreasing order of their modified release dates $\bar{r}_j$; as a tie-breaking criterion, we consider non-increasing order of $w_j/p_j$. The resulting

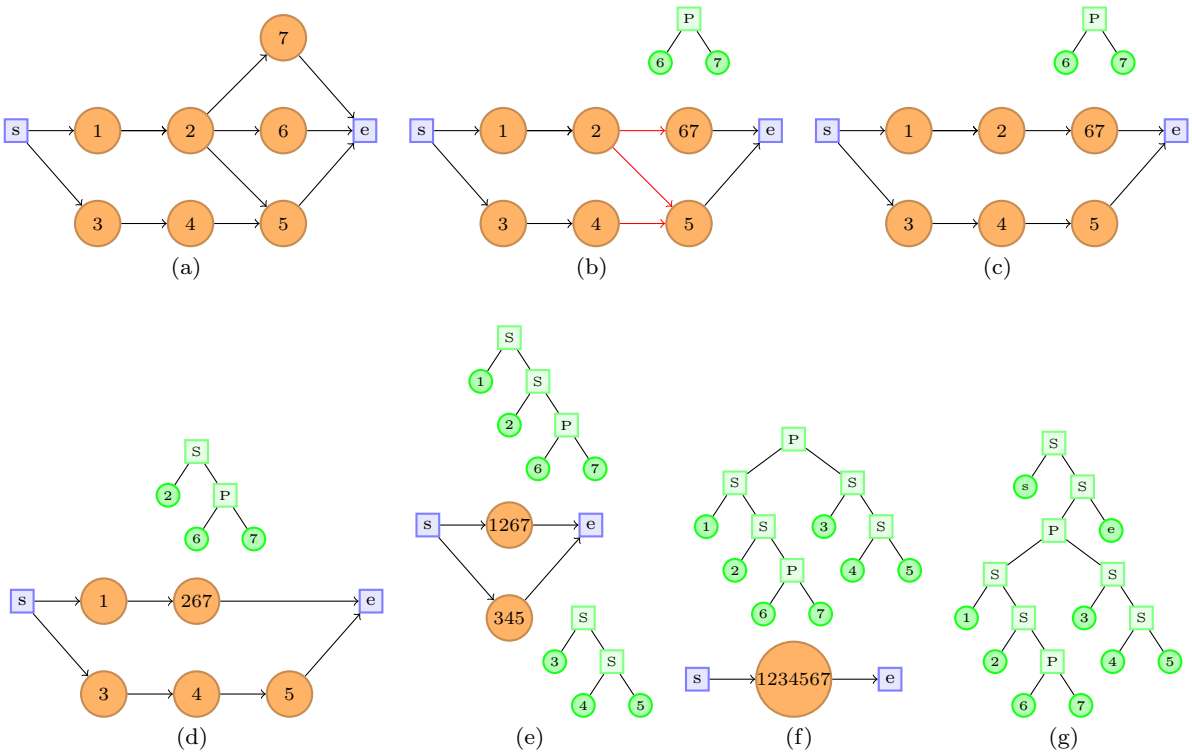


**Fig. 7:** Modified traversal algorithm applied to the input graph in (a).

*non-delay* sequence of jobs is given by $\sigma^r = (1, \ldots, n)$, where a sequence is said to be 'non-delay' if the machine is never kept idle while some jobs are waiting to be processed [34]. Let $B_i = (u_i, \ldots, v_i)$ be one block (in which jobs are sorted according to their new indices). The set $B = \{B_1, \ldots, B_\kappa\}$ is a *valid* decomposition of the job set into $\kappa$ blocks if the following conditions are satisfied:

1. $u_1 = 1$;
2. for each $i, j$ with $1 < i \leq \kappa$ and $1 \leq j \leq n$, if $u_i = j$ then $v_{i-1} = j - 1$ and vice versa;
3. for each $i, j$ with $1 \leq i \leq \kappa$ and $u_i \leq j \leq v_i$, we have $\bar{r}_{u_i} + \sum_{s=u_i}^{j-1} p_s \geq \bar{r}_j$.

Although the sequencing of the jobs within one block is actually still open, the sequencing of the blocks is predetermined. Given a valid set of blocks $B$, we compute $\mathrm{LB}_1$ for each block $B_i \in B$ separately. The value $\mathrm{LB}_2$ is then the sum of the bounds per block; analogously to [18, 33, 37], $\mathrm{LB}_2$ can be shown to be a lower bound for P.

We define $\mathrm{LB}_1^* = L_1(\lambda^*, \mu^*)$, where $\lambda^*$ and $\mu^*$ are optimal choices for the Lagrangian multipliers for $\mathrm{LB}_1$, and $\mathrm{LB}_2^*$, which is computed by adding the contribution $L_1(\lambda_{B_i}^*, \mu_{B_i}^*)$ for each block $B_i$, where $\lambda_{B_i}^*$ and $\mu_{B_i}^*$ are the optimal choices for the multipliers for block $B_i$.

**Theorem 5** $\mathrm{LB}_1^* \leq \mathrm{LB}_2^*$.

Although TPA might not find $\lambda_{B_i}^*$ and $\mu_{B_i}^*$ and thus the same result as Theorem 5 might not hold for $\mathrm{LB}_1$ and $\mathrm{LB}_2$, empirical results show that $\mathrm{LB}_2$ is on average far tighter than $\mathrm{LB}_1$ (these results are shown in Table 8).

*Job splitting* It sometimes happens that the decomposition procedure fails to improve the bound (only one block is created and $\mathrm{LB}_2 = \mathrm{LB}_1$). Another approach is to explicitly re-introduce the release-date constraints (which have been removed previously). We define problem $\mathrm{P}_2$, which is a generalization of $\mathrm{P}_1$ in which the release-date constraints (19) are included. The associated Lagrangian problem is:

$$\mathrm{LR}_{\mathrm{P}_2}: \quad L_2(\lambda, \mu) = \min \sum_{j \in N} w'_j C_j + c$$

subject to constraints (19),(22)–(24).

Contrary to $\mathrm{LR}_{\mathrm{P}_1}$, we now remove the serial-parallel precedence constraints because they render the Lagrangian problem too difficult. Problem $\mathrm{LR}_{\mathrm{P}_2}$ is a total-weighted-completion-times problem with release dates. This problem is known to be NP-hard [28], but a number of efficient polynomial algorithms, which are based

on job splitting, have been proposed to compute tight lower bounds [7,18,31]. One of these algorithms is the SS procedure proposed by Belouadah et al. [7], which runs in $O(n \log n)$ time and which we adopt here. Essentially, we again decompose the job set into a set of blocks $B$ and compute $L_2(\lambda, \mu)$ for each block $B_i \in B$. The lower bound $LB_2^{SS_r}$ is again the sum of the contributions of the individual blocks. Experiments show that $LB_2^{SS_r}$ is typically tighter than $LB_2$ when the release dates are unequal. With equal release dates, on the other hand, normally $LB_2 \geq LB_2^{SS_r}$ because $LB_2$ incorporates a part of the precedence graph. TPA is applied also here for multiplier updates.

We introduce $P'_2$, which is a generalization of $P_1$ where deadline constraints are retrieved by inclusion of the constraint set (20). The associated Lagrangian problem is:

$$LR_{P'_2}: \quad L'_2(\lambda, \mu) = \min \sum_{j \in N} w'_j C_j + c$$
$$\text{subject to constraints (20),(22)–(24).}$$

$LR_{P'_2}$ is a total-weighted-completion-times problem with deadlines. This problem is known to be NP-hard [28]. Posner [35] proposes a job-splitting lower bounding scheme for $LR_{P'_2}$ that uses $O(n \log n)$ time; the lower bound $LB_2^{SS_\delta}$ results from block decomposition and computation of $L'_2(\lambda, \mu)$ for each block. We again apply TPA for setting the multiplies.

### *5.3.5 Improvement by slack variables*

Relaxed inequality constraints can be considered to be 'nasty' constraints because they decrease the quality of lower bounds. We follow Hoogeveen and van de Velde [20] in exploiting the advantages of slack variables to lessen the effect of such nasty constraints to improve the quality of the lower bounds.

We introduce two non-negative vectors of slack variables: vector $\mathbf{y} = (y_1, \ldots, y_n)$ and vector $\mathbf{z} = (z_{11}, \ldots, z_{1n}, \ldots, z_{n1}, \ldots, z_{nn})$. Consider the following sets of constraints:

$$T_j = C_j - d_j + y_j \qquad \forall j \in N \qquad (27)$$
$$C_j = C_i + p_j + z_{ij} \qquad \forall (i,j) \in A \qquad (28)$$
$$y_j, z_{ij} \geq 0 \qquad \forall i,j \in N \qquad (29)$$

Let problem $P_3$ be the variant of problem $P_1$ in which the sets of constraints (18) and (21) are replaced by the constraints (27)–(29). The Lagrangian problem associated with $P_3$ is:

$$LR_{P_3}: \quad L_3(\lambda, \mu) = \min \sum_{j=1}^{n} (w_j - \lambda_j) T_j + \sum_{j=1}^{n} \lambda_j y_j +$$
$$\sum_{j=1}^{n} \sum_{k \in \mathcal{Q}_j} \mu_{jk} z_{jk} + \sum_{j \in N} w'_j C_j + c$$
$$\text{subject to constraints (22)–(25) and (29).}$$

The values of the variables $T_j$, $y_j$ and $z_{jk}$ are zero in any optimal solution to $LR_{P_3}$ because for $i,j \in N$ the following inequalities hold: $0 \leq \lambda_j \leq w_j$ and $\mu_{jk} \geq 0$. In an optimal solution to $P_3$, however, these values might not be zero. In fact, according to the set of constraints (27), unless $C_j = d_j$, either $T_j$ or $y_j$ is nonzero. Also, from constraints (28), $z_{jk}$ may not be zero when job $j$ has at least two successors or job $k$ has at least two predecessors in $G$. We introduce three problems that each carry a part of the objective function of $LR_{P_3}$, one of which is $LR_{P_1}$ and the other two are the following two slack-variable (SV) problems, where $Y$ is the set of all $\mathbf{y}$-vectors corresponding to feasible solutions to $P_3$ and $Z$ similarly contains all $\mathbf{z}$-vectors.

$$P_{SV1}: \quad SV_1(\lambda) = \min \sum_{j=1}^{n} (w_j - \lambda_j) T_j + \sum_{j=1}^{n} \lambda_j y_j$$
$$\text{subject to constraints (22),(23),(25) and } \mathbf{y} \in Y;$$

$$P_{SV2}: \quad SV_2(\mu) = \min \sum_{j=1}^{n} \sum_{k \in \mathcal{Q}_j} \mu_{jk} z_{jk}$$
$$\text{subject to constraint } \mathbf{z} \in Z.$$

Note that the term $\sum_{j=1}^{n} (w_j - \lambda_j) T_j$ appears in two of the problems, but it will be set to zero anyway in $LR_{P_1}$.

Hoogeveen and van de Velde [20] propose $O(n \log n)$-time procedures to compute valid lower bounds for $P_{SV1}$ and $P_{SV2}$. Let $LB_{SV1} \geq 0$ and $LB_{SV2} \geq 0$ be lower bounds for $P_{SV1}$ and $P_{SV2}$, respectively. By adding $LB_{SV1}$ and $LB_{SV2}$ to $LB_2$, a better lower bound $LB_3$ for P is obtained [20]. The same SV problems can also be constructed for $LB_2^{SS_r}$ and $LB_2^{SS_\delta}$ to lead to bounds $LB_3^{SS_r} = LB_2^{SS_r} + LB_{SV1} + LB_{SV2}$ and $LB_3^{SS_\delta} = LB_2^{SS_\delta} + LB_{SV1} + LB_{SV2}$. We have the following result:

**Observation 1** $LB_2 \leq LB_3$, $LB_2^{SS_r} \leq LB_3^{SS_r}$ *and* $LB_2^{SS_\delta} \leq LB_3^{SS_\delta}$.

### *5.3.6 Other Lagrangian bounds*

All the lower bounds introduced in this section are based on the formulation (17)-(24). Other Lagrangian-relaxation-based lower bounds have also been proposed for special cases of this problem. These other bounds are

mostly based on other (conceptual) formulations. For example, to achieve a lower bound, Lagrangian penalties could be added to the objective function while allowing jobs to be processed repeatedly. Many variants of such a lower bound exist [42–44], but most of these variants are either too weak or too slow. Another lower bound based on Lagrangian relaxation is obtained by relaxing the capacity constraints, such that jobs are allowed to be processed in parallel in exchange for Lagrangian penalties [45].

## 6 Dominance properties

Our search procedure also incorporates a number of dominance rules, which will be described in this section. We will use the following additional notation. Given two partial sequences $\pi = (\pi_1, \ldots, \pi_k)$ and $\pi' = (\pi'_1, \ldots, \pi'_{k'})$, we define a merge operator as follows: $\pi|\pi' = (\pi_1, \ldots, \pi_k, \pi'_1, \ldots, \pi'_{k'})$. If $\pi'$ contains only one job $j$ then we can also write $\pi|j = (\pi_1, \ldots, \pi_k, j)$, and similarly if $\pi = (j)$ then $j|\pi' = (j, \pi'_1, \ldots, \pi'_{k'})$.

### 6.1 General dominance rules

We use the lower bounds proposed in Section 5 to prune the search tree. Let $\mathrm{LB}(U)$ represent any of the lower bounds described in Section 5, applied to the set $U$ of unscheduled jobs, and let $S_{best}$ be the currently best known feasible solution. Notice that $\mathrm{TWT}(S_{best})$ is an upper bound for $\mathrm{TWT}(S^*)$. The following dominance rule is then immediate:

**Dominance rule 1 ($\mathbf{DR_1}$)** *Consider a node associated with selection $S_P$. If*

$$\mathrm{TWT}(S_P) + \mathrm{LB}(U) \geq \mathrm{TWT}(S_{best}),$$

*then the node associated with $S_P$ can be fathomed.*

As we already introduced in Section 4, a partial schedule can be denoted by either $S_P$ or $(\sigma_B, \sigma_E)$. Multiple lower bounds can be used to fathom a node. The selection of lower bounds and the order in which they are computed, obviously influences the performance of the B&B algorithm. These issues are examined in Section 8.2.

The subset of *active schedules* is dominant for total weighted tardiness problems [10, 34]. A feasible schedule is called *active* if it is not possible to construct another schedule by changing the sequence of jobs such that at least one job is finishing earlier and no other job finishes later. The dominance of active schedules holds even when deadlines and precedence constraints are given.

**Dominance rule 2 ($\mathbf{DR_2}$)** *Consider a node associated with $(\sigma_B, \emptyset)$ that is selected for forward branching, and let $j$ be a job belonging to $E_B$. If $\bar{r}_j \geq \min_{k \in E_B}\{\bar{r}_k + p_k\}$, then the child node associated with the schedule $(\sigma_B|j, \emptyset)$ can be fathomed.*

We also prune a branch whenever an obvious violation of the deadline constraints is detected. A partial schedule associated with a particular node is not always extended to a feasible schedule. Scheduling a job in one particular position may force other jobs to violate their deadline constraints, even though it does not violate its own constraints. Let $\mathcal{A}$ be an arbitrary subset of $U$ and let $\Pi_{\mathcal{A}}$ be the set of all possible permutations of jobs in $\mathcal{A}$. The following theorem states when a job is scheduled in a 'wrong position', meaning that it will lead to a violation of deadline constraints.

**Theorem 6** *Consider a partial schedule $(\sigma_B, \sigma_E)$. If there exists any non-empty subset $\mathcal{A} \subset U$ such that the inequality $\min_{\pi \in \Pi_{\mathcal{A}}}\{C_{\max}(\sigma_B|\pi)\} > \max_{j \in \mathcal{A}}\{\bar{\delta}_j\}$ holds, then the schedule $(\sigma_B, \sigma_E)$ is not feasible.*

The problem $\min_{\pi \in \Pi_{\mathcal{A}}}\{C_{\max}(\sigma_B|\pi)\}$, which equates with $1|r_j, \delta_j, prec|C_{\max}$, is NP-hard because the mere verification of the existence of a feasible schedule is already NP-complete. We remove deadlines and create a new problem whose optimal solution is computed in $\mathrm{O}(n^2)$ time [25]. For computational efficiency, we use a linear-time lower bound for this new problem. This lower bound is computed as follows: $\min_{j \in \mathcal{A} \cap E_B}\{\bar{r}_j\} + \sum_{j \in \mathcal{A}} p_j$.

**Dominance rule 3 ($\mathbf{DR_3}$)** *The node associated with $(\sigma_B, \sigma_E)$ can be eliminated if at least one of the following conditions is satisfied:*

1. *if $\sigma_E = \emptyset$ and the condition of Theorem 6 is satisfied for the partial schedule $(\sigma_B, \emptyset)$;*
2. *if $\sigma_E \neq \emptyset$ and $\max_{j \in U}\{\bar{\delta}_j\} < st(\sigma_E)$.*

While additional precedence constraints could be added to the problem considering time windows and using constraint propagation techniques, the solution representation for our B&B algorithms and the above dominance rules ($\mathrm{DR}_2$ and $\mathrm{DR}_3$) are devised in such a way that any violation of these additional precedence constraints is dominated. Consider two jobs $i$ and $j$ with $p_i = p_j = 10$, $r_i = 0$, $r_j = 5$, $\delta_i = 20$ and $\delta_j = 30$. Using constraint propagation techniques, it could be possible to include an extra constraint that allows the processing of job $j$ to occur only after the completion of job $i$. Such an additional constraint is not necessary, however, because in the above-described situation, all sequences in which job $j$ precedes job $i$ will be automatically fathomed by $\mathrm{DR}_2$ and $\mathrm{DR}_3$. Moreover, increasing

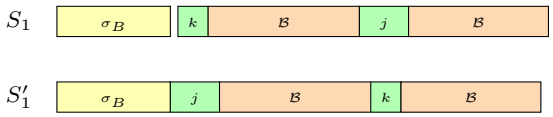


**Fig. 8:** Schedules $S_1$ and $S'_1$.

the density of the precedence graph in this way would also decrease the tightness of the lower bounds, which is undesirable.

## 6.2 Dominance rule based on two-job interchange

We describe a dominance rule based on job interchange. This dominance rule consists of two parts. The first part deals with the interchange of jobs in an FB node whereas the second part deals with the interchange of jobs in a BB node.

### *6.2.1 Interchanging jobs in an FB node*

In an FB node, consider jobs $j, k \in E_B$ that are not identical (they differ in at least one of their parameters). We will always assume that $\hat{r}_k < \hat{r}_j + p_j$ and $\hat{r}_j < \hat{r}_k + p_k$, because otherwise Dominance rule 2 enforces the scheduling of the job with smaller $\hat{r}$ before the job with larger $\hat{r}$; note here that $\hat{r}_j = \bar{r}_j$ and $\hat{r}_k = \bar{r}_k$ because all predecessors of jobs $j$ and $k$ has already been scheduled and therefore the branching decisions cover the propagation of precedence constraints. We also assume that any successor of job $k$ is also a successor of job $j$ ($\mathcal{Q}_k \subset \mathcal{Q}_j$). Consider a node of the search tree in which job $k$ is scheduled at or after the completion of sequence $\sigma_B$. Suppose that the partial schedule associated to the current node can be extended to a feasible schedule $S_1$ in which job $j$ is scheduled somewhere after job $k$. We define a set $\mathcal{B} = U \backslash \{j, k\}$ of jobs. We also construct a schedule $S'_1$ by interchanging jobs $j$ and $k$ while the order of jobs belonging to $\mathcal{B}$ remains unchanged. Figure 8 illustrates schedules $S_1$ and $S'_1$.

To prove that interchanging jobs $j$ and $k$ does not increase the total weighted tardiness, we argue that the gain of interchanging jobs $j$ and $k$, which is computed as $\mathrm{TWT}(S_1) - \mathrm{TWT}(S'_1)$, is greater than or equal to zero, no matter when job $j$ is scheduled. Let $st_j(S)$ denote the start time of job $j$ in schedule $S$. Remember that $st(\pi)$ denotes the start time of a sequence $\pi$. Let $\tau_1$ be the difference between the start time of job $j$ in $S_1$ and the start time of $k$ in $S'_1$. If $st_k(S'_1)$ is less than $st_j(S_1)$ then $\tau_1$ is negative, otherwise it is non-negative. By interchanging jobs $j$ and $k$ each job that belongs to set $\mathcal{B}$ may be shifted either to the right or to the left. Let $\tau_2 \geq 0$ be the maximum shift to the right of the jobs belonging to set $\mathcal{B}$. Notice that if all jobs in $\mathcal{B}$ are shifted to the left, then $\tau_2 = 0$. For each $t$ as the start time of job $j$ in $S_1$, Jouglet et al. [22] define a function $\Gamma_{jk}(t, \tau_1, \tau_2)$ as follows:

$$\begin{aligned}\Gamma_{jk}(t, \tau_1, \tau_2) = {} & w_j \max\{0, t + p_j - d_j\} \\ & - w_k \max\{0, t + \tau_1 + p_k - d_k\} \\ & + w_k \max\{0, \hat{r}_k + p_k - d_k\} \\ & - w_j \max\{0, \hat{r}_j + p_j - d_j\} - \tau_2 \sum_{i \in \mathcal{B}} w_i.\end{aligned}$$

For the sub-problem of P where precedence and deadline constraints are removed, Jouglet et al. [22] show that $\Gamma_{jk}(t, \tau_1, \tau_2)$ is a lower bound for the gain of interchanging jobs $j$ and $k$ when $t = st_j(S_1)$. This result can be improved by adding the gain of shifting the jobs which are tardy in both schedules $S_1$ and $S'_1$. We introduce the set $\mathcal{B}'$ of jobs where each job $i \in \mathcal{B}'$ is certainly a tardy job in $S'_1$. Let $\hat{\mathcal{P}}_i$ be the set of transitive predecessors of job $i$. The following set of jobs, which is a subset of $\mathcal{B}'$, is used in our implementations because the order based on which the jobs in $\mathcal{B}$ are scheduled has not yet been defined and therefore computing $\mathcal{B}'$ is not possible:

$$\left\{ i \in \mathcal{B} \middle| \hat{r}_j + p_j + \sum_{l \in (\mathcal{B} \cap \hat{\mathcal{P}}_i)} p_l + p_i \geq d_i \right\}.$$

Let $\tau'_2 \geq 0$ be the minimum shift to the left of the jobs belonging to set $\mathcal{B}$. Note that at least one of the values $\tau'_2$ and $\tau_2$ equals zero. We define the function $\hat{\Gamma}_{jk}(t, \tau_1, \tau_2, \tau'_2)$ as follows:

$$\hat{\Gamma}_{jk}(t, \tau_1, \tau_2, \tau'_2) = \Gamma_{jk}(t, \tau_1, \tau_2) + \tau'_2 \sum_{i \in \mathcal{B}'} w_i.$$

The values $\tau_2$ and $\tau'_2$ cannot be negative. Therefore, we immediately infer $\Gamma_{jk}(t, \tau_1, \tau_2) \leq \hat{\Gamma}_{jk}(t, \tau_1, \tau_2, \tau'_2)$. We need the following result:

**Theorem 7** *$\hat{\Gamma}_{jk}(t, \tau_1, \tau_2, \tau'_2)$ is a valid lower bound for the gain of interchanging jobs $j$ and $k$.*

In a general setting (problem P), however, job interchanges are not always feasible for every starting time $t$. We opt for verifying the feasibility of an interchange by ensuring that it does not cause any violation of deadlines and/or precedence constraints for all possible $t = st_j(S_1)$. Let $\Psi$ be an upper bound for the completion time of the sequence $S'_1$, computed as follows:

$$\Psi = \max\left\{ \max\{\hat{r}_j + p_j, \hat{r}_k\} + p_k, \max_{i \in \mathcal{B}}\{\hat{r}_i\} \right\} + \sum_{i \in \mathcal{B}} p_i.$$

The following theorem provides the conditions under which for every possible $t = st_j(S_1)$ interchanging jobs $j$ and $k$ is feasible.

**Theorem 8** *For each feasible schedule $S_1$, an alternative feasible schedule $S_1'$ is created by interchanging jobs $j$ and $k$, if the following conditions are satisfied:*

1. $\bar{\delta}_j - p_j \leq \bar{\delta}_k - \tau_1 - p_k$ *or* $\Psi \leq \hat{\delta}_k$;
2. $\tau_2 = 0$ *or* $\Psi \leq \min_{i \in \mathcal{B}}\{\hat{\delta}_i\}$.

Jouglet et al. [22] prove that if $w_j \geq w_k$ then the value $\Gamma_{jk}(\max\{d_j - p_j, \hat{r}_k + p_k\}, \tau_1, \tau_2)$ is the minimum gain obtained by interchanging jobs $j$ and $k$ for the setting where deadlines and precedence constraints are removed. We derive a more general result using the following lemma.

**Lemma 4** *Let $f : t \to \alpha \max\{0, t-a\} - \beta \max\{0, t-b\} + C$ be a function defined on $[u, v]$ for $a, b, C \in \mathbb{R}$ and $\alpha, \beta, u, v \in \mathbb{R}^+$. The function $f$ reaches a global minimum at value $t^*$ computed as follows:*

$$t^*(\alpha, \beta, a, b, u, v) = \begin{cases} \min\{\bar{u}, v\} & \text{if } \alpha \geq \beta \\ u & \text{if } \alpha < \beta,\ b > a,\ \alpha(\bar{v} - \bar{u}) \geq \beta(\bar{v} - b) \\ v & \text{otherwise} \end{cases}$$

*where $\bar{u} = \max\{u, a\}$ and $\bar{v} = \max\{v, b\}$.*

Corollary 1 below follows from Theorem 7, Theorem 8 and Lemma 4, if we choose $\alpha = w_j$, $\beta = w_k$, $a = d_j - p_j$, $b = d_k - \tau_1 - p_k$, $u = \hat{r}_k + p_k$, $v = \delta_j - p_j$ and $C = w_k \max\{0, \hat{r}_k + p_k - d_k\} - w_j \max\{0, \hat{r}_j + p_j - d_j\} - \tau_2 \sum_{i \in \mathcal{B}} w_i + \tau_2' \sum_{i \in \mathcal{B}'} w_i$. Let $st_j^*$ be computed as follows:

$$st_j^* = t^*(w_j, w_k, d_j - p_j, d_k - \tau_1 - p_k, \hat{r}_k + p_k, \delta_j - p_j).$$

**Corollary 1** *$\Gamma_{jk}^*(\tau_1, \tau_2, \tau_2') = \hat{\Gamma}_{jk}(st_j^*, \tau_1, \tau_2, \tau_2')$ is the minimum gain obtained by interchanging jobs $j$ and $k$, provided that for every possible $st_j(S_1)$ interchanging jobs $j$ and $k$ is feasible.*

To compute $\Gamma_{jk}^*(\tau_1, \tau_2, \tau_2')$, the values of $\tau_1$, $\tau_2$ and $\tau_2'$ must be known. We establish an exhaustive list of cases for which $\tau_1$, $\tau_2$ and $\tau_2'$ can be computed, which is summarized in Table 4. Given a particular case, the values $\tau_1$, $\tau_2$ and $\tau_2'$ are computed as follows:

**Table 4** Interchange cases.

| Case | $(\hat{r}_j + p_j - \hat{r}_k - p_k)$ | $(p_j - p_k)$ | $(\max_{i \in U}\{\hat{r}_i\} - \hat{r}_k - p_k)$ | $(\hat{r}_j - \hat{r}_k)$ | $(\max_{i \in U}\{\hat{r}_i\} - \hat{r}_k - p_j)$ |
|---|---|---|---|---|---|
| 1 | $\leqslant 0$ | $< 0$ | $\geqslant 0$ | - | - |
| 2 | $\lessgtr 0$ | $< 0$ | $< 0$ | $\leqslant 0$ | $> 0$ |
| 3 | $\lessgtr 0$ | $< 0$ | $< 0$ | $\lessgtr 0$ | $\leqslant 0$ |
| 4 | $\lessgtr 0$ | $< 0$ | $< 0$ | $> 0$ | - |
| 5 | $\lessgtr 0$ | $\geqslant 0$ | $\geqslant 0$ | - | - |
| 6 | $\lessgtr 0$ | $\geqslant 0$ | $< 0$ | - | - |
| 7 | $> 0$ | $< 0$ | - | - | - |
| 8 | $> 0$ | $\geqslant 0$ | - | - | - |

$$\tau_1 = \begin{cases} 0 & \text{Cases 1,5} \\ \max_{i \in U}\{\hat{r}_i\} - \hat{r}_k - p_k & \text{Case 2} \\ \max\{\hat{r}_j + p_j, \max_{i \in \mathcal{B}}\{\hat{r}_i\}\} - \hat{r}_k - p_k & \text{Cases 3,4,6} \\ \hat{r}_j + p_j - \hat{r}_k - p_k & \text{Cases 7,8} \end{cases}$$

$$\tau_2 = \begin{cases} p_k - p_j & \text{Case 1} \\ \max_{i \in U}\{\hat{r}_i\} - \hat{r}_k - p_j & \text{Case 2} \\ 0 & \text{Cases 3,5,6} \\ \max\{\hat{r}_j + p_j, \max_{i \in \mathcal{B}}\{\hat{r}_i\}\} - \hat{r}_k - p_j & \text{Case 4} \\ \hat{r}_j - \hat{r}_k & \text{Case 7} \\ \hat{r}_j + p_j - \hat{r}_k - p_k & \text{Case 8} \end{cases}$$

$$\tau_2' = \begin{cases} 0 & \text{Cases 1,2,4,5,7,8} \\ \hat{r}_k - \hat{r}_j & \text{Case 3} \\ \hat{r}_k + p_k - \max\{\hat{r}_j + p_j, \max_{i \in \mathcal{B}}\{\hat{r}_i\}\} & \text{Case 6} \end{cases}$$

Following the above results, the first part of Dominance rule 4 is derived.

**Dominance rule 4 (DR$_4$; first part)** *Given an FB node associated with $(\sigma_B, \emptyset)$, if there exist two non-identical jobs $j, k \in E_B$ with $\mathcal{Q}_k \cap \mathcal{Q}_j = \mathcal{Q}_k$ and the inequality $\Gamma_{jk}^*(\tau_1, \tau_2, \tau_2') > 0$ holds, then $(\sigma_B|j, \emptyset)$ dominates $(\sigma_B|k, \emptyset)$.*

### *6.2.2 Interchanging jobs in a BB node*

Let $j, k \in E_E$ where jobs $j$ and $k$ are not identical. We also assume that any unscheduled predecessor of job $k$ is also a predecessor of job $j$. In other words, we have $\mathcal{P}_k \cap \mathcal{P}_j \cap U = \mathcal{P}_k \cap U$. Consider a BB node of the search tree with decision job $k$. The partial schedule associated with the current node can be extended to a feasible schedule $S_2$ in which job $j$ is scheduled before job $k$ but after all jobs in the sequence $\sigma_B$. The set $\mathcal{B}$ is the set of all remaining unscheduled jobs where $\mathcal{B} = U \backslash \{j, k\}$. Let schedule $S_2'$ be constructed by interchanging jobs $j$ and $k$ while keeping the order based on which the jobs

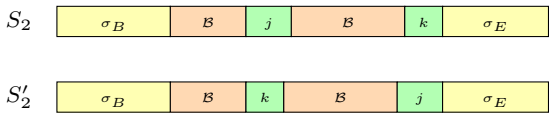


**Fig. 9:** Schedules $S_2$ and $S'_2$.

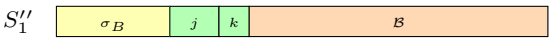


**Fig. 10:** Schedule $S''_1$.

belonging to $\mathcal{B}$ will be scheduled. Figure 9 illustrates schedules $S_2$ and $S'_2$.

For each $t$ as the start time of job $j$ in $S_2$, we define a function $\Delta_{jk}(t)$ as follows:

$$\begin{aligned}\Delta_{jk}(t) = {} & w_j \max\{0, t + p_j - d_j\} \\ & - w_k \max\{0, t + p_k - d_k\} \\ & + w_k \max\{0, st(\sigma_E) - d_k\} \\ & - w_j \max\{0, st(\sigma_E) - d_j\} - \max\{0, p_k - p_j\} \sum_{i \in \mathcal{B}} w_i.\end{aligned}$$

In a BB node, for each $t$ as the start time of job $j$, $\Delta_{jk}(t)$ is a lower bound of the gain of interchanging jobs $k$ and $j$, if the conditions of Theorem 9 are satisfied. Theorem 9 provides the conditions on which for every possible $t = st_j(S_1)$ interchanging jobs $j$ and $k$ is feasible.

**Theorem 9** *For each feasible schedule $S_2$, a feasible schedule $S'_2$ can be created by interchanging jobs $j$ and $k$, if the following conditions are satisfied:*

1. $st(\sigma_E) \leq \hat{\delta}_j$;
2. $p_k - p_j \leq 0$ *or* $st(\sigma_E) - p_j \leq \min_{i \in \mathcal{B}} \hat{\delta}_i$.

Corollary 2 follows from Theorem 9 and Lemma 4, if we choose $\alpha = w_j$, $\beta = w_k$, $a = d_j - p_j$, $b = d_k - p_k$, $u = C_{\max}(\sigma_B)$, $v = st(\sigma_E) - p_k - p_j$ and $C = w_k \max\{0, st(\sigma_E) - d_k\} - w_j \max\{0, st(\sigma_E) - d_j\} - \max\{0, p_k - p_j\} \sum_{i \in \mathcal{B}} w_i$. Let $st_j^{*\prime}$ be computed as follows:

$$\begin{aligned}st_j^{*\prime} = t^*(& w_j, w_k, d_j - p_j, d_k - p_k, \\ & C_{\max}(\sigma_B) + \sum_{i \in \mathcal{P}_j \cap U} p_i, st(\sigma_E) - p_k - p_j).\end{aligned}$$

**Corollary 2** *$\Delta^*_{jk} = \Delta_{jk}(st_j^{*\prime})$ is the minimum gain obtained by interchanging jobs $j$ and $k$, provided that for every possible $t = st_j(S_1)$ interchanging jobs $j$ and $k$ is feasible.*

Following the above results, the second part of Dominance rule 4 is derived.

**Dominance rule 4 (DR$_4$; second part)** *Given a BB node associated with $(\sigma_B, \sigma_E)$, if there exist two non-identical jobs $j, k \in E_E$ with $\mathcal{P}_k \cap \mathcal{P}_j \cap U = \mathcal{P}_k \cap U$ and $\Delta^*_{jk} > 0$, then $(\sigma_B, j|\sigma_E)$ dominates $(\sigma_B, k|\sigma_E)$.*

### 6.3 Dominance rule based on job insertion

We describe a dominance rule based on job insertion. This dominance rule, similar to the dominance rule based on job interchange, consists of two parts. The first part deals with the insertion of a job in an FB node whereas the second part deals with the insertion of a job in a BB node.

#### *6.3.1 Inserting a job in an FB node*

In an FB node, let $j, k \in E_B$ where jobs $j$ and $k$ are not identical. Again we assume that $\hat{r}_k < \hat{r}_j + p_j$ and $\hat{r}_j < \hat{r}_k + p_k$, otherwise Dominance rule 2 enforces scheduling the job with smaller $\hat{r}$ before the job with larger $\hat{r}$ (remind that $\hat{r}_j = \bar{r}_j$ and $\hat{r}_k = \bar{r}_k$ because all predecessors of jobs $j$ and $k$ have already been scheduled and therefore the branching decisions cover precedence constraints propagation). Consider an FB node of the search tree in which job $k$ is scheduled after the jobs in sequence $\sigma_B$. Assume that the partial schedule associated with the current node can be extended to the feasible schedule $S_1$ depicted in Figure 8. We construct a schedule $S''_1$ by inserting the job $j$ before job $k$ while keeping the order of jobs belonging to $\mathcal{B}$. Figure 10 illustrates the construction of the schedule $S''_1$.

Let $\tau_3$ be the maximum shift to the right of the jobs belonging to $\mathcal{B}$, which is computed as follows:

$$\tau_3 = \max\left\{0, \hat{r}_j + p_j + p_k - \max\left\{\hat{r}_k + p_k, \min_{i \in \mathcal{B}}\{\bar{r}_i\}\right\}\right\}.$$

For each $t$ as the start time of job $j$ in schedule $S_1$, we define a function $\Gamma'_{jk}(t, \tau_3)$ as follows:

$$\begin{aligned}\Gamma'_{jk}(t, \tau_3) = {} & w_j \max\{0, t + p_j - d_j\} \\ & - w_k \max\{0, \hat{r}_j + p_j + p_k - d_k\} \\ & + w_k \max\{0, \hat{r}_k + p_k - d_k\} \\ & - w_j \max\{0, \hat{r}_j + p_j - d_j\} - \tau_3 \sum_{i \in OJ} w_i.\end{aligned}$$

Job insertion, similar to job interchange, is not always feasible for every starting time $t$ of job $j$. We verify feasibility of an insertion by ensuring that it does not cause any deadline and/or precedence-constraint violation for all possible $t = st_j(S_1)$. Let $\Psi'$ be an upper

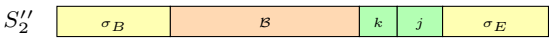


**Fig. 11:** Schedule $S_2''$.

bound for the completion time of the sequence $S_1'$, computed as follows:

$$\Psi' = \max\left\{\hat{r}_j + p_j + p_k, \max_{i\in\mathcal{B}}\{\hat{r}_i\}\right\} + \sum_{i\in\mathcal{B}} p_i.$$

The following theorem provides the conditions under which for every possible $t = st_j(S_1)$ inserting job $j$ before job $k$ is feasible.

**Theorem 10** *For each feasible schedule $S_1$, another feasible schedule $S_1''$ can be created by inserting job $j$ before job $k$ if the following conditions hold:*

1. $\hat{r}_j + p_j + p_k \leq \hat{\delta}_k$;
2. $\tau_3 = 0$ *or* $\Psi' \leq \min_{i\in\mathcal{B}}\{\hat{\delta}_i\}$.

Corollary 3 below follows from Theorem 10.

**Corollary 3** $\Gamma'^{*}_{jk}(\tau_3) = \Gamma'_{jk}(\hat{r}_k + p_k, \tau_3) = \Gamma_{jk}(\hat{r}_k + p_k, \hat{r}_j + p_j - \hat{r}_k - p_k, \tau_3)$ *is the minimum gain obtained by inserting job $j$ before job $k$ provided that for every possible $t = st_j(S_1)$ inserting job $j$ before job $k$ is feasible.*

Following the above results, the first part of Dominance rule 5 is derived.

**Dominance rule 5 (DR$_5$; first part)** *Consider an FB node associated with $(\sigma_B, \emptyset)$. If there exist two non-identical jobs $j, k \in E_B$ for which the inequality $\Gamma'^{*}_{jk}(\tau_3) > 0$ holds, then $(\sigma_B|j, \emptyset)$ dominates $(\sigma_B|k, \emptyset)$.*

### *6.3.2 Inserting a job in a BB node*

In a BB node, let $j, k \in E_E$ where jobs $j$ and $k$ are not identical. Consider a node of the search tree in which job $k$ is scheduled before sequence $\sigma_E$. Assume that the partial schedule associated with the current node can be extended to the feasible schedule $S_2$ depicted in Figure 9. We also construct a schedule $S_2''$ by inserting the job $j$ to be scheduled after job $k$ but before the jobs in the sequence $\sigma_E$ and by keeping the order of jobs belonging to $\mathcal{B}$. Figure 11 illustrates schedule $S_2''$.

For each $t$, which is the start time of job $j$ in schedule $S_2$, we define the function $\Delta'_{jk}(t)$ as follows:

$$\begin{aligned}\Delta'_{jk}(t) = {} & w_j \max\{0, t + p_j - d_j\} \\ & - w_k \max\{0, st(\sigma_E) - p_j - d_k\} \\ & + w_k \max\{0, st(\sigma_E) - d_k\} \\ & - w_j \max\{0, st(\sigma_E) - d_j\}.\end{aligned}$$

Similarly to the previous results, for each feasible schedule $S_2$, a feasible schedule $S_2''$ is constructed by inserting jobs $j$ after job $k$, if $st(\sigma_E) \leq \hat{\delta}_j$. The following corollary is obtained:

**Corollary 4** $\Delta'^{*}_{jk} = \Delta'_{jk}(C_{\max}(\sigma_B) + \sum_{i\in\mathcal{P}_j\cap U} p_i)$ *is the minimum gain obtained by inserting job $j$ after job $k$ provided that $st(\sigma_E) \leq \hat{\delta}_j$.*

Following the above results, the second part of Dominance rule 5 is derived.

**Dominance rule 5 (DR$_5$; second part)** *Consider a BB node associated with $(\sigma_B, \sigma_E)$. If there exist two non-identical jobs $j, k \in E_E$ for which the inequality $\Delta'^{*}_{jk} > 0$ holds, then $(\sigma_B, j|\sigma_E)$ dominates $(\sigma_B, k|\sigma_E)$.*

## 6.4 Dominance rules on scheduled jobs

The *dominance theorem of dynamic programming* (see Jouglet et al. [22]) is another existing theorem that can be used to eliminate nodes in the search tree. It compares two partial sequences that contain identical subsets of jobs and eliminates the one having the larger total weighted tardiness. When total weighted tardiness values are the same, then only one of the sequences is kept. Let us consider two feasible partial sequences $\sigma_1$ and $\sigma_2$ ($\sigma_2$ is a feasible permutation of $\sigma_1$) of $k$ jobs, where $k < n$. Let $\mathcal{C}$ be the set of jobs in either $\sigma_1$ or $\sigma_2$. We are going to decide whether it is advantageous to replace $\sigma_2$ by $\sigma_1$ in all (partial) schedules in which $\sigma_2$ orders the last $k$ jobs. The set of scheduled jobs and the set of unscheduled jobs are identical for both $\sigma_1$ and $\sigma_2$. Sequence $\sigma_1$ is as good as sequence $\sigma_2$ if it fulfills one of the following conditions:

1. $C_{\max}(\sigma_1) \leq C_{\max}(\sigma_2)$ and $\mathrm{TWT}(\sigma_1) \leq \mathrm{TWT}(\sigma_2)$;
2. $C_{\max}(\sigma_1) > C_{\max}(\sigma_2)$ and the following inequality also holds:

$$\mathrm{TWT}(\sigma_1) + (\min_{i\in U}\{\bar{r}_i^{\sigma_1}\} - \min_{i\in U}\{\bar{r}_i^{\sigma_2}\}) \sum_{i\in U} w_i \leq \mathrm{TWT}(\sigma_2),$$

where $\bar{r}_i^{\sigma_1}$ is the updated release date associated with the sequence $\sigma_1$ and $\bar{r}_i^{\sigma_2}$ is the updated release date associated with the sequence $\sigma_2$.

Jouglet et al. [22] determine the sequences that can be replaced by a dominant permutation. They find that sequence $\sigma_1$ dominates sequence $\sigma_2$ if the following two conditions hold:

1. sequence $\sigma_1$ is as good as sequence $\sigma_2$;
2. sequence $\sigma_2$ is not as good as $\sigma_1$ or $\sigma_1$ has lexicographically smaller release dates than $\sigma_2$.

Note that the second condition enforces a tie-breaking rule where a lexicographical number associated to each sequence is computed and among those sequences that are equivalent, the one with lower lexicographic number is selected. To avoid conflicts with Dominance rule 2, jobs are renumbered in non-decreasing order of their release dates $r_j$.

**Dominance rule 6 ($DR_6$)** *If there exists a better feasible permutation of $\sigma_B$ and/or a better feasible permutation of $\sigma_E$, then the node $(\sigma_B, \sigma_E)$ is fathomed.*

If $\sigma_E = \emptyset$ and there is a better feasible permutation of $\sigma_B$, then the dominance is proven similarly to Theorem 13.6 in [22]. If $\sigma_E \neq \emptyset$, then all jobs belonging to the set $U$ will be scheduled between $C_{\max}(\sigma_B)$ and $st(\sigma_E) = C_{\max}(\sigma_B) + \sum_{j \in U} p_j$. Therefore, all permutations of $\sigma_E$ start at time $st(\sigma_E)$ and if there exists at least one better feasible permutation of $\sigma_E$, then fathoming the node associated with $(\sigma_B, \sigma_E)$ does not eliminate the optimal solution.

Dominance rule 6 where only permutations of the last $k$ jobs are considered, is referred to as $DR_6^k$. Computing $DR_6^n$ amounts to enumerating all $O(n!)$ feasible solutions, which would yield an optimal solution but is computationally prohibitive. In our B&B algorithm, we therefore choose $k < n$. There is a trade-off between the computational effort needed to compute $DR_6^k$ and the improvement achieved by eliminating dominated nodes. Based on initial experiments (see Table 5; more details on the instance generation are provided in Section 8.1), we observe that the algorithms perform worse when $k > 6$. We also notice that it is not efficient to use $DR_6^k$ when $k > |U|$ because the computational effort to solve the subproblem consisting of the remaining $|U|$ jobs is less than the computational effort needed to enumerate all feasible permutations of the last $k$ jobs. Thus, $k = \min\{|\sigma_B|, |U|, 6\}$ while scheduling forward and $k = \min\{|\sigma_E|, |U|, 6\}$ when scheduling backward.

We observe that in BB2 with unequal release dates, at certain moments during the search procedure, we switch from forward to backward branching, which forces us to start with $k = |\sigma_E| = 0$ and we thus lose a number of pruning opportunities.

## 7 Initial upper bound

Although for most of the instances the B&B algorithm finds a reasonably good solution (a tight upper bound) quickly, there are instances for which feasible solutions are encountered only after a large part of the search tree has been scanned. Therefore, we initialize the upper bound in the root node of the B&B algorithm using a stand-alone (heuristic) procedure, which we refer to as *time-window heuristic* (TWH).

**Table 5** Average CPU times (in seconds; first number) and number of unsolved instances within the time limit (between brackets, if any; out of 864) for different choices of $k$ in BB1 run on Ins.

| $k$ | $n$ | | | |
|---|---|---|---|---|
| | 20 | 30 | 40 | 50 |
| 2 | 0.0043 | - | - | - |
| 3 | **0.0038** | 0.045 | - | - |
| 4 | 0.0039 | **0.039** | 5.157 (1) | - |
| 5 | 0.0042 | 0.039 | 3.499 | 15.895 (15) |
| 6 | 0.0043 | 0.041 | **3.130** | **13.358 (13)** |
| 7 | 0.0050 | 0.046 | 4.741 (1) | 14.301 (15) |
| 8 | - | 0.092 | 7.459 (1) | 22.470 (17) |

**Algorithm 1** Time-window heuristic (TWH)

**Input:** a sequence $\sigma$

**for** $itr = 1$ to $2$ **do**
  IMPROVE_BY_SWAP
  $k = 0$
  **while** $k \leq 50$ **do**
    **if** $k$ even **then**
      $start = 0$
    **else**
      $start = \min\{minsize, maxsize/2\}$
    **end if**
    **while** $start + minsize \leq n$ **do**
      $end = \min\{start + maxsize, n\}$
      SP $\leftarrow$ CONST_SP($\sigma, start, end$)
      $\sigma_{\text{SP}} \leftarrow$ SOLVE_BB(SP)
      $\sigma' \leftarrow$ COPYSEQ($\sigma, \sigma_{\text{SP}}, start, end$)
      **if** TWT($\sigma'$) < TWT($\sigma$) **then**
        $\sigma \leftarrow \sigma'$
      **end if**
    **end while**
    $k = k + 1$
  **end while**
**end for**

**Output:** TWT($\sigma$)

The key idea of our TWH is to iteratively locally improve a given sequence of jobs within a varying time window (Algorithm 1); similar ideas have already been proposed in the literature [11, 24]. It starts with any given sequence (note that finding a feasible sequence might be very difficult for some instances, so we also allow infeasible sequences). Then to locally improve the solution, the algorithm constructs a number of subproblems. Each subproblem is defined by two positions: a *start* position and an *end* position. The subproblem tries to optimally resequence the jobs that are positioned between the given *start* and *end* positions in the initial sequence such that the completion time of the subsequence does not exceed the start time of the job in the position $end + 1$. This additional condition is fulfilled by updating the deadline of all jobs $j$ in the subproblem to $\delta_j^{SP} = \min\{\delta_j, st_{end+1}(\sigma)\}$ and updat-

ing the release date of all jobs $j$ in the subproblem to $r_j^{SP} = \max\{r_j, C_{start-1}(\sigma)\}$.

In TWH, the subprocedure IMPROVE_BY_SWAP is a naive local search procedure in which each pair of jobs is examined for swapping exactly once, in a steepest descent fashion. The length of the subsequence to be reoptimized is in between $minsize = 10$ and $maxsize = \min\{\max\{n/2, 10\}, 20\}$. Given a *start* and an *end* position, CONST_SP constructs the associated subproblem and SOLVE_BB solves the subproblem using the same branch-and-bound algorithms explained in this paper. A new sequence is constructed using COPY-SEQ.

The input sequence for TWH is the result of a dynamic priority rule that stepwise schedules jobs (from time 0 onwards) that are eligible according to the precedence constraints (meaning that all predecessors were previously already selected) and whose release date has already been reached; if no job is eligible in this way, then the algorithm proceeds to the earliest ready time of all jobs for which all predecessors have already been scheduled. If multiple jobs are eligible, then priority is given to the one with the earliest deadline and the lowest processing time. In the computation of the eligible set of jobs the deadline constraints are ignored. Therefore, the resulting sequence might not be feasible to P. In such cases, we add a large infeasibility cost to the objective function in the hope of finding a feasible solution during TWH.

The upper bound that is the output of TWH improves the runtime for those instances for which the branch-and-bound algorithm fails to find a feasible solution fast. Furthermore, this upper bound turns out to be optimal for most of the instances of P and for those instances for which the optimal solution is not found, the optimality gap is very low; see Table 6 (see Section 8.1 for more details on the different instance sets). To evaluate the efficiency of TWH, we have run some computational experiments, the results of which are reported in Table 6 and Table 7. In Table 6, column *total* contains the total number of instances and the values in column *feas* represent the number of instances for which at least one feasible solution exists. Column *fnd* reports the number of times TWH finds a feasible solution, column *opt* counts the number of optimal solutions found and column *gap* states the average optimality gap, averaged only for the instances for which the optimal solution was not found by TWH. The average optimality gap is computed as the average value of ((UB − OPT)/UB), with UB the output of TWH. Table 7 reports the average CPU times for the same subset of instances studied in Table 6. Note that the column that reports the average CPU times for $\text{Ins}^{\text{PAN}}$ pertains to all instances with $n = 30, 40$ and 50.

**Table 6** The performance of TWH

| Instance Set | | total | feas | TWH | | |
|---|---|---|---|---|---|---|
| | | | | fnd | opt | gap |
| $\text{Ins}^{L}$ | $n = 30$ | 432 | 401 | 397 | 300 | 0.012 |
| | $n = 40$ | 432 | 395 | 392 | 313 | 0.011 |
| | $n = 50$ | 432 | 397 | 395 | 302 | 0.025 |
| $\text{Ins}^{\text{PAN}}$ | $n = 30$ | 100 | 100 | 100 | 59 | 0.003 |
| | $n = 40$ | 100 | 100 | 100 | 72 | 0.001 |
| | $n = 50$ | 94* | 94 | 89 | 50 | 0.001 |
| $\text{Ins}^{\text{TAN}}$ | $n = 40$ | 875 | 875 | 875 | 542 | 0.019 |
| | $n = 50$ | 875 | 875 | 875 | 545 | 0.015 |

∗ the optimal solutions are only available for 94 instances.

**Table 7** Average CPU times (in seconds) of upper bound computation for different instance sets

| $\text{Ins}^{L}$ | | | $\text{Ins}^{\text{PAN}}$ | $\text{Ins}^{\text{TAN}}$ | |
|---|---|---|---|---|---|
| $n = 30$ | $n = 40$ | $n = 50$ | | $n = 40$ | $n = 50$ |
| 0.04 | 0.09 | 0.17 | 0.08 | 0.31 | 0.66 |

## 8 Computational results

All algorithms have been implemented in VC++ 2010, while CPlex 12.3 is used to solve the MIP formulations. All computational results were obtained on a laptop Dell Latitude with 2.6 GHz Core(TM) i7-3720QM processor, 8GB of RAM, running under Windows 7.

### 8.1 Instance generation

To the best of our knowledge, there are no benchmark sets of instances of problem P available, and so we have generated our own set of instances, which is referred to as Ins. Two sets of benchmark instances for subproblems of P are also used in our experiments; these are referred to as $\text{Ins}^{\text{TAN}}$ and $\text{Ins}^{\text{PAN}}$ and are discussed in Section 8.5.1 and 8.5.2, respectively.

The set Ins consists of the two disjoint subsets $\text{Ins}^{S}$ and $\text{Ins}^{L}$): $\text{Ins}^{S}$ contains instances with small processing times and $\text{Ins}^{L}$ holds instances with large processing times. The values $p_i$ $(1 \leq i \leq n)$ are sampled from the uniform distribution $U[1, \alpha]$, where $\alpha = 10$ for $\text{Ins}^{S}$ and $\alpha = 100$ for $\text{Ins}^{L}$. For each subset, we generate instances with $|N| = n = 10$, 20, 30, 40 and 50 jobs. Release dates $r_i$ are drawn from $U[0, \tau P]$, where $P = \sum_{i \in N} p_i$ and $\tau \in \{0.0, 0.5, 1.0\}$. Due dates $d_i$ are generated from $U[r_i + p_i, r_i + p_i + \rho P]$ with $\rho \in \{0.05, 0.25, 0.50\}$ and weights $w_i$ stem from $U[1, 10]$. Up to here our generation is based on the instance generation procedure of Tanaka and Fujikuma [42]. Our modifications pertain to the generation of deadlines and precedence relations

**Table 8** Average percentage gap from optimal value.

| | | $LB_0$ | $LB_1$ | $LB_2$ | $LB_2^{SS_r}$ | $LB_2^{SS_\delta}$ | $LB_3$ | $LB_3^{SS_r}$ | $LB_3^{SS_\delta}$ | $LB_{Best}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $OS$ | 0.00 | 50.505 | 50.505 | 44.369 | 43.313 | 36.857 | 43.142 | 42.086 | 35.630 | 35.372 |
| | 0.25 | 67.776 | 63.465 | 53.444 | 52.702 | 52.300 | 51.955 | 51.013 | 50.568 | 49.834 |
| | 0.50 | 71.461 | 66.108 | 52.378 | 51.890 | 52.021 | 51.216 | 50.727 | 50.767 | 50.352 |
| | 0.75 | 77.055 | 69.836 | 50.769 | 50.520 | 50.565 | 49.430 | 49.182 | 49.041 | 48.863 |
| $\tau$ | 0.00 | 38.141 | 29.169 | 29.169 | 29.182 | 24.784 | 28.152 | 28.165 | 23.667 | 23.667 |
| | 0.50 | 76.712 | 73.157 | 59.554 | 58.724 | 57.699 | 57.876 | 57.046 | 55.922 | 55.656 |
| | 1.00 | 85.704 | 85.539 | 62.494 | 61.255 | 61.655 | 61.117 | 59.878 | 60.233 | 59.309 |
| All | - | 67.161 | 62.911 | 50.638 | 49.950 | 48.268 | 49.275 | 48.586 | 46.824 | 46.425 |

among jobs. Deadlines are chosen from $U[d_i, d_i + \phi P]$ with $\phi \in \{1.00, 1.25, 1.50\}$.

The addition of precedence constraints may lead to the generation of many instances with no feasible solution. For this reason, for each instance we first construct a feasible solution without considering precedence constraints (using branch-and-bound). Next, the jobs are re-indexed according to the job order in this feasible solution. If no feasible solution exists even without precedence constraints, we use the original indices. Subsequently, a precedence graph is created using the RanGen software [12] with $OS \in \{0.00, 0.25, 0.50, 0.75\}$, where $OS$ is the *order strength* of the graph (a measure for the density of the graph). For any instance, if a feasible solution exists without precedence constraints, then the addition of precedence constraints will never render it infeasible because RanGen only generates arcs from lower-indexed to higher-indexed jobs.

In conclusion, for each combination of $(\alpha, n, \tau, \rho, \phi, OS)$, four instances are generated; the total number of instances is thus $2 \times 5 \times 3 \times 3 \times 3 \times 4 \times 4 = 4320$. In all our experiments, the time limit is set to 1200 seconds. If an instance is not solved to guaranteed optimality, it is said to be 'unsolved' for the procedure. Throughout this section, we report averages computed only over the solved instances.

## 8.2 Lower bounds

We compare the quality of the lower bounds for the subset of instances with large processing times and $n = 30$. We set $k_{\max} = 10$ for all lower bounds. The detailed results of this comparison are reported in Table 8.

The average gap for $LB_1$ is less than or equal to that for $LB_0$, especially when the precedence graph is dense; for $OS = 0$, on the other hand, there are no precedence constraints and $LB_0$ and $LB_1$ are essentially the same. A similar observation can be made for $LB_1$ and $LB_2$, where the gap for $LB_2$ is noticeably smaller than that for $LB_1$ when release dates are imposed, while in the case $\tau = 0$, only one block is created and the lower bounds $LB_1$ and $LB_2$ coincide. The average gap for $LB_3$ is indeed smaller than that for $LB_2$, as was to be expected according to Observation 1.

Although we have no theoretical result that would indicate a better performance of $LB_2^{SS_r}$ in comparison with $LB_2$, the average gap for $LB_2^{SS_r}$ is less than that for $LB_2$ in case of non-zero release dates. When release dates are zero, however, the gap for $LB_2^{SS_r}$ is larger than or equal to that for $LB_2$. In fact, when release dates are zero, only one block is created and constraints (19) can be removed from $LR_{P_2}$, and thus $LR_{P_2}$ is a relaxation of $LR_{P_1}$. $LB_2^{SS_\delta}$ performs better than $LB_2$ and $LB_2^{SS_r}$ for most of the instances. Since $LB_2^{SS_\delta}$ and $LB_2^{SS_r}$ require the same computational effort, we decide not to use $LB_2^{SS_r}$ in our B&B algorithms, where enumeration of child nodes is more efficient than extra computation of a weak bound. The gap for $LB_3^{SS_r}$ is less than that for $LB_2^{SS_r}$ and a similar observation holds for $LB_3^{SS_\delta}$ versus $LB_2^{SS_\delta}$, which confirms the result in Observation 1. Again, since $LB_3^{SS_r}$ and $LB_3^{SS_\delta}$ are equally expensive in terms of computational effort, we decide not to use $LB_3^{SS_r}$.

In our final implementation, we will not compute all the bounds for all the nodes because this consumes too much effort. We start with computing $LB_T$, $LB_0$ and $LB_{SV1}$ for the unscheduled jobs. Let $S_{best}$ be the best feasible schedule found. If the node is fathomed by $DR_1$, then we backtrack; otherwise if $TWT(S_P) + \lceil LB_0 + LB_{SV1} \rceil \times 1.4 < TWT(S_{best})$ then we do not compute the remaining lower bounds and continue branching. If the latter equality does not hold, then we anticipate that with a better bound we might still be able to fathom the node, and we compute $LB_3$ and/or $LB_3^{SS_\delta}$. For all lower bounds we choose $k_{\max} = 0$ if $OS < 0.5$ and $k_{\max} = 1$ otherwise.

## 8.3 Dominance rules

In each node of the B&B algorithm, dominance rules are tested. Based on some preliminary experiments, we find that applying the rules in the following order performs

**Table 9** The list of scenarios.

| Scenario | $DR_2$ | $DR_3$ | $DR_6^2$ | $DR_4$ | $DR_5$ | $DR_6^k$ | $DR_1$ |
|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | - | - | - | - |
| 2 | ✓ | ✓ | ✓ | ✓ | - | - | - |
| 3 | ✓ | ✓ | ✓ | ✓ | ✓ | - | - |
| 4 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - |
| 5 | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ |
| 6 | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ |
| 7 | ✓ | ✓ | - | ✓ | ✓ | - | ✓ |
| 8 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

**Table 10** The effect of the dominance rules.

| Method | Scenario | $n = 20$ CPU | $n = 20$ Nodes | $n = 30$ CPU | $n = 30$ Nodes |
|---|---|---|---|---|---|
| BB1 | 1 | 0.004 | 12521 | - | - |
| | 2 | 0.003 | 4310 | 2.956 | 4388157 |
| | 3 | 0.003 | 4279 | 3.547 | 4382728 |
| | 4 | 0.004 | 839 | 0.260 | 48410 |
| | 5 | 0.008 | 1912 | 0.075 | 10095 |
| | 6 | 0.003 | 488 | 0.016 | 3442 |
| | 7 | 0.772 | 1044361 | - | - |
| | 8 | 0.003 | 487 | 0.039 | 3451 |
| BB2 | 1 | 0.003 | 11698 | - | - |
| | 2 | 0.002 | 3609 | 1.506 | 2182869 |
| | 3 | 0.002 | 3271 | 1.482 | 1669982 |
| | 4 | 0.003 | 980 | 0.260 | 91985 |
| | 5 | 0.004 | 1658 | 0.135 | 30474 |
| | 6 | 0.002 | 490 | 0.055 | 9750 |
| | 7 | 0.155 | 239389 | - | - |
| | 8 | 0.002 | 427 | 0.047 | 7464 |

'-' means that the implementation fails to solve many instances within the time limit.

well, and we will therefore follow this order throughout the algorithm:

$$DR_2, DR_3, DR_6^2, DR_4, DR_5, DR_6^k, DR_1.$$

In order to evaluate the effectiveness of the rules, we examine a number of scenarios with respect to the selection of the implemented bounds; the list of scenarios is given in Table 9. Scenario 1 includes the simplest combination of dominance rules, namely $DR_2$, $DR_3$ and $DR_6^2$. From Scenario 2 to Scenario 4, extra rules are gradually added. In Scenario 5, all dominance rules are active except $DR_4$, and in Scenario 6, only $DR_5$ is inactive. Scenario 7 similarly includes all dominance rules except $DR_6$. Finally, in Scenario 8, all dominance rules are active.

For each of these implementations, we report the average CPU times and the average number of nodes explored in the search tree in Table 10; the results pertain to the instances of Ins with $n = 20, 30$. Scenarios 2 and 3 show the effect of $DR_4$ and $DR_5$. In Scenario 2, $DR_4$ improves the performance of both algorithms whereas in Scenario 3, $DR_5$ has a beneficial effect only for BB2. Scenario 4 reflects the impact of $DR_6$ for $k$ jobs.

Comparing Scenario 5 to Scenario 8, we see that inclusion of $DR_1$ has a strong beneficial effect on both algorithms; the effect is strongest in BB2 because tighter bounds can be computed by scheduling backward. From Table 10, we learn that apart from $DR_2$, which is always crucial in total tardiness scheduling problems, the most important dominance rule is $DR_6$: deactivating this rule triggers a huge increase in the average number of nodes and the average CPU times; incorporating $DR_4$ also has a marked effect (compare Scenarios 5 and 8). Among all dominance rules tested, $DR_5$ is the least important; removing $DR_5$ slightly increases the node count and the runtimes in BB2. In BB1, removing $DR_5$ even decreases the number of nodes and the runtimes; it turns out that for $n > 30$, however, the effect of $DR_5$ is also (slightly) beneficial for BB1, and so we decide to adopt Scenario 8 as the final setting in which the experiments in the following sections will be run.

As a side note, we observe that for all the foregoing dominance rules, after the root node, omitting the precedence constraints implied by sets $\mathcal{Q}_j$ and $\mathcal{P}_j$ from the updates of $\bar{r}_j$ and $\bar{\delta}_j$ has only little effect. We will therefore not include these precedence constraints into the updated release dates and deadlines and thus avoid the additional computational overhead.

**Table 11** Average CPU times (in seconds) and number of unsolved instances within the time limit (out of 432) for the MIP formulations and the B&B algorithms run on Ins with $n = 10$, 20 and 30.

| $\alpha$ | Method | $n$ = 10 | $n$ = 20 | $n$ = 30 |
|---|---|---|---|---|
| 10 | ASF | 0.81 | – | – |
| | ASF′ | 0.80 | – | – |
| | TIF | 0.43 | 2.02 | 53.47 (3) |
| | TIF′ | 0.64 | 2.97 | 88.17 (12) |
| | BB1 | 0.00 | 0.00 | 0.02 |
| | BB2 | 0.00 | 0.00 | 0.03 |
| 100 | ASF | 0.92 | – | – |
| | ASF′ | 0.95 | – | – |
| | TIF | 6.54 | – | – |
| | TIF′ | 21.78 | – | – |
| | BB1 | 0.00 | 0.00 | 0.06 |
| | BB2 | 0.00 | 0.00 | 0.06 |

## 8.4 Branch-and-bound algorithms

In this section we discuss the performance of our B&B algorithms. In Table 11 we compare the performance of BB1 and BB2 with the MIP formulations discussed in Section 3. In this table as well as in the following, we report the average runtime and the number of unsolved instances (if there are any).

Based on Table 11, we conclude that the time-indexed formulations are far better than the assignment formulations when processing times are small. For large

**Table 12** Average CPU times (in seconds) and number of unsolved instances within the time limit (out of 432) for BB1 and BB2 run on Ins with $n = 40$ and 50.

| $\alpha$ | Method | $n$ | |
|---|---|---|---|
| | | 40 | 50 |
| 10 | BB1 | 1.26 | 16.99 (1) |
| | BB2 | 1.65 | 35.73 (6) |
| 100 | BB1 | 5.00 | 14.00 (12) |
| | BB2 | 3.38 (1) | 18.28 (12) |

**Table 13** Average CPU times (in seconds) and number of unsolved instances within the time limit (out of 126) for different choices of $n$ and $OS$ in BB1 and BB2 run on Ins.

| Method | $n$ | $OS$ | | | |
|---|---|---|---|---|---|
| | | 0 | 0.25 | 0.50 | 0.75 |
| BB1 | 30 | 0.08 | 0.04 | 0.02 | 0.01 |
| | 40 | 11.85 | 0.55 | 0.10 | 0.02 |
| | 50 | 50.81 (13) | 12.63 | 0.68 | 0.06 |
| BB2 | 30 | 0.05 | 0.10 | 0.03 | 0.01 |
| | 40 | 7.37 (1) | 2.37 | 0.32 | 0.03 |
| | 50 | 44.17 (12) | 57.12 (6) | 8.66 | 0.10 |

processing times, the performance of ASF is slightly better than TIF. Although ASF′ and TIF′ are tighter than their counterparts with aggregate precedence constraints, the extra computational effort needed to process the larger models increases CPU times in both TIF′ and ASF′. The B&B algorithms BB1 and BB2 both clearly outperform the MIP formulations regardless of the size of the processing times. Table 12 shows the performance of BB1 and BB2 applied to the larger instances of Ins ($n = 40$ and 50) that cannot be solved by the MIP formulations. On average, BB1 performs better than BB2, although this does not hold for all parameter settings (more details follow below). BB1 solves all instances with 40 jobs and fails to solve around 1.5% of the instances with 50 jobs. BB2 fails to solve one instance with 40 jobs and around 2% of the instances with 50 jobs. We will indicate below that all these unsolved instances belong to a specific class; it is worth mentioning that the difficult instances are not the same for the two B&B algorithms.

The number of precedence constraints obviously affects the performance of the algorithms. On the one hand, by adding precedence constraints, the set of feasible sequences shrinks; on the other hand, the lower bounds also become less tight. The net result of these two effects is a priori not predictable. For instance classes without release dates and deadlines ($r_j = 0$ and $\delta_j = \infty$), the quality of the lower bound is very good when $OS = 0$, therefore the effect of a weaker bound due to higher $OS$ will be more pronounced than when release dates and deadlines are also imposed.

To identify the classes of difficult instances, we focus on case $n = 50$. Table 14 shows the outcomes of the experiments for each combination of $\tau$, $\rho$ and $OS$. According to this table, the most time-consuming class of instances is the one where release dates are neither loose nor tight ($\tau = 0.50$), due dates are loose ($\rho = 0.50$) and the set of precedence constraints is empty ($OS = 0$). No clear pattern can be distinguished for the algorithmic performance as a function of the tightness of the deadlines, so these results are excluded from the table. The unsolved instances are distributed differently for the two algorithms, although $\tau = 0.5$ in all and $OS = 0$ in most of the unsolved instances. For example, BB1 solves all instances with $OS = 0.25$ whereas BB2 does not solve six of these instances. Also, BB1 fails to solve two instances with $\rho = 0.25$ whereas this occurs for only one instance with $\rho = 0.25$ for BB2.

We will represent each class of instances by a triple $(\tau, \rho, OS)$. As mentioned before, the hardest class of instances for both algorithms is $(0.5, 0.5, 0)$. The class $(0, 0.5, 0)$, which seems to be the third most difficult class for BB1, is very easy for BB2. Also, $(0.5, 0.5, 0.25)$, which is the second hardest class for BB2, does not require very high runtimes from BB1. We infer that BB2 is better than BB1 when release dates are equal (zero); in this case (cf. Section 4) stronger bounds are computed in backward scheduling. Conversely, BB1 is better than BB2 when release dates are not equal (especially when $\tau = 0.50$). With unequal release dates, backward branching cannot start from the root node but rather only after a certain number of jobs have already been scheduled. Because branching forward increases the earliest possible starting and completion times of jobs, the trivial lower bound and the Lagrangian-based lower bounds will be stronger for BB1 than those for BB2. As explained at the end of Section 6.4 and contrary to BB2, in BB1 $k$ is never restarted in the computation of $DR_6^k$ and therefore we do not lose any pruning opportunity.

## 8.5 Experiments for subproblems of P

In this section, we present the results of our B&B algorithms for subproblems of P that have also been studied in earlier literature.

### *8.5.1 A single-machine problem with precedence constraints:* $1|prec|\sum w_jT_j$

One special case of P is single machine scheduling with precedence constraints where the objective is to minimize the total weighted tardiness. From our observations in Section 8.4, we know that we only need to

**Table 14** Average CPU times (in seconds) and number of unsolved instances within the time limit (out of 24) for different choices of $\tau$, $\rho$ and $OS$ in BB1 and BB2 run on Ins with $n = 50$.

| Method | $\tau$ | $\rho$ | $OS$ | | | |
|---|---|---|---|---|---|---|
| | | | 0 | 0.25 | 0.50 | 0.75 |
| BB1 | 0 | 0.05 | 0.27 | 0.50 | 0.18 | 0.03 |
| | | 0.25 | 26.84 | 11.89 | 0.51 | 0.05 |
| | | 0.50 | 127.59 | 25.92 | 1.96 | 0.06 |
| | 0.5 | 0.05 | 1.60 | 5.16 | 0.44 | 0.05 |
| | | 0.25 | 35.16 (2) | 8.86 | 0.72 | 0.07 |
| | | 0.50 | 439.09 (11) | 57.55 | 1.86 | 0.09 |
| | 1 | 0.05 | 1.77 | 0.44 | 0.15 | 0.04 |
| | | 0.25 | 1.14 | 0.50 | 0.16 | 0.05 |
| | | 0.50 | 0.49 | 2.86 | 0.16 | 0.06 |
| BB2 | 0 | 0.05 | 0.21 | 1.01 | 0.25 | 0.03 |
| | | 0.25 | 0.27 | 2.79 | 0.36 | 0.05 |
| | | 0.50 | 0.49 | 3.11 | 0.37 | 0.06 |
| | 0.5 | 0.05 | 1.66 | 76.47 | 5.97 | 0.12 |
| | | 0.25 | 77.95 (1) | 158.78 | 17.96 | 0.20 |
| | | 0.50 | 544.25 (11) | 338.02 (6) | 52.42 | 0.30 |
| | 1 | 0.05 | 1.75 | 0.43 | 0.15 | 0.06 |
| | | 0.25 | 1.09 | 0.51 | 0.29 | 0.05 |
| | | 0.50 | 0.47 | 3.15 | 0.19 | 0.05 |

consider BB2 for this subproblem because all release dates are zero, and so we compare the performance of BB2 with the SSDP algorithm proposed by Tanaka and Sato [44]. We apply both algorithms to the benchmark instances $\text{Ins}^{\text{TAN}}$ obtained from [44]. For these instances, parameter $Pr$ denotes the probability that each arc $(i, j) \in N \times N$ with $i \neq j$ is present in the precedence graph. Note that the resulting precedence graph may contain transitive arcs. In such cases, the transitive reduction is computed and used as input to BB2. Table 15 shows the computational results for our B&B algorithms and for the SSDP algorithm (which was run on the same computer). SSDP solves instances in very short runtimes when there are no precedence constraints. SSDP performs worse, however, when the precedence graph is dense, while the B&B algorithms will tend to perform better exactly in this case. To conclude this comparison, we underline the fact that our algorithms have been developed to solve the more general setting in which time windows are also imposed, whereas the instance set examined here does not contain such time windows.

#### *8.5.2 A single machine problem with time windows:* $1|r_j, \delta_j| \sum w_j C_j$

Another special case of P is the single machine problem with time windows where the objective function is to minimize the total weighted sum of the completion times. We run our B&B algorithms on one of the instance sets provided by Pan and Shi [33], which has been introduced as problem set (I) in which the parameters $\alpha$ and $\beta$ define the ranges for the generation of release dates and deadlines, respectively. We refer to this instance set as $\text{Ins}^{\text{PAN}}$. To solve these instances, we set all due dates to zero. Table 16 shows the computational results of BB1 and BB2 applied to $\text{Ins}^{\text{PAN}}$. Our B&B algorithms both solve 394 out of the 400 instances to optimality within the time limit of 1200 seconds. Although a consistent pattern cannot be recognized, it seems that the hardest instances belong to the subsets where $\alpha = 1$ and $\beta = 16$. Since we do not have access to the code of Pan and Shi, direct comparisons are difficult, but overall our runtimes are of the same order of magnitude, although the most difficult instances for Pan and Shi are not the most difficult ones for our code, and vice versa.

**Table 15** Average CPU times (in seconds) for different choices of $Pr$ and $n$ in BB2 and SSDP run on $\text{Ins}^{\text{TAN}}$.

| $Pr$ | $n = 40$ | | $n = 50$ | |
|---|---|---|---|---|
| | BB2 | SSDP | BB2 | SSDP |
| 0 | 0.04 | 0.04 | 0.26 | 0.06 |
| 0.005 | 0.49 | 0.35 | 1.83 | 0.71 |
| 0.01 | 0.61 | 0.51 | 4.98 | 2.05 |
| 0.02 | 0.80 | 1.67 | 15.79 | 6.40 |
| 0.05 | 1.48 | 6.01 | 32.11 | 37.13 |
| 0.10 | 0.57 | 9.71 | 2.64 | 32.78 |
| 0.20 | 0.09 | 1.67 | 0.18 | 3.61 |

Contrary to the discussion in Section 8.4, we notice that our two algorithms behave quite similarly for these instances. This can be explained as follows. First, for all members of $\text{Ins}^{\text{PAN}}$, release dates are non-zero, such that BB2 follows the same steps as BB1 until the release dates of all remaining jobs are less than the decision time. Second, the fact that due dates are zero makes all jobs late already in the root node and thus the scheduling of any job (even in the beginning of the schedule) has a positive contribution in the objective value. For the case where due dates are non-zero, scheduling backward is advantageous because jobs are mostly early in the beginning of the schedule, so they have zero contribution in the objective value.

## 9 Summary and conclusion

In this article, we have developed exact algorithms for the single-machine scheduling problem with total weighted tardiness penalties. We work with a rather general problem statement, in that both precedence constraints as well as time windows (release dates and deadlines) are part of the input; this generalizes quite a number of problems for which computational pro-

**Table 16** Average CPU times (in seconds; first number) and number of unsolved instances within the time limit (between brackets, if any; out of 10) for different choices of $n$, $\alpha$ and $\beta$ in BB1 and BB2 run on $\text{Ins}^{\text{PAN}}$.

| $n$ | $\alpha$ | $\beta$ | Method | |
|---|---|---|---|---|
| | | | BB1 | BB2 |
| 20 | 0.5 | 1 | 0.006 | 0.005 |
| | | 2 | 0.008 | 0.005 |
| | | 4 | 0.010 | 0.008 |
| | | 8 | 0.012 | 0.010 |
| | | 16 | 0.012 | 0.010 |
| | 1 | 1 | 0.002 | 0.002 |
| | | 2 | 0.002 | 0.002 |
| | | 4 | 0.004 | 0.004 |
| | | 8 | 0.013 | 0.011 |
| | | 16 | 0.009 | 0.008 |
| 30 | 0.5 | 1 | 0.026 | 0.022 |
| | | 2 | 0.040 | 0.047 |
| | | 4 | 0.055 | 0.053 |
| | | 8 | 0.100 | 0.110 |
| | | 16 | 0.105 | 0.107 |
| | 1 | 1 | 0.007 | 0.008 |
| | | 2 | 0.033 | 0.033 |
| | | 4 | 0.017 | 0.018 |
| | | 8 | 0.161 | 0.160 |
| | | 16 | 0.092 | 0.087 |
| 40 | 0.5 | 1 | 0.111 | 0.123 |
| | | 2 | 0.306 | 0.301 |
| | | 4 | 1.419 | 1.451 |
| | | 8 | 2.512 | 2.477 |
| | | 16 | 0.987 | 0.967 |
| | 1 | 1 | 0.014 | 0.012 |
| | | 2 | 0.234 | 0.165 |
| | | 4 | 0.161 | 0.170 |
| | | 8 | 0.267 | 0.282 |
| | | 16 | 0.867 | 0.875 |
| 50 | 0.5 | 1 | 0.782 | 0.784 |
| | | 2 | 3.028 | 3.038 |
| | | 4 | 11.082 | 11.152 |
| | | 8 | 17.801 | 17.718 |
| | | 16 | 100.041 | 100.253 |
| | 1 | 1 | 0.036 | 0.038 |
| | | 2 | 1.660 | 1.638 |
| | | 4 | 13.466(1) | 13.686(1) |
| | | 8 | 71.697(2) | 72.407(2) |
| | | 16 | 77.316(3) | 77.406(3) |

cedures have already been published. We develop a branch-and-bound algorithm that solves the problem to guaranteed optimality. Computational results show that our approach is effective in solving medium-sized instances, and that it compares favorably with two straightforward linear formulations. Our procedure was also compared with two existing methods, namely an SSDP algorithm and a B&B algorithm, for special cases of the problem. The SSDP algorithm requires only very low runtimes in the absence of precedence constraints, but it performs worse when the precedence graph is dense, which is exactly the easiest setting for our B&B algorithms.

**Acknowledgments**

We are very grateful to professor Shunji Tanaka from Kyoto University for providing us with the benchmark instances and the code of the SSDP algorithm that were used in Section 8.5.1, and to professor Yunpeng Pan from South Dakota State University for sending us the instances examined in Section 8.5.2.

## Appendix: Proofs

**Proof of Lemma 1:** Consider the set of constraints (10). For each $(i,j) \in A$, the following inequalities hold:

$$\begin{aligned}
&x_{i1} + \cdots + x_{in} \leq 1 - x_{j1} = x_{j2} + \cdots + x_{jn} \\
&x_{i2} + \cdots + x_{in} \leq 1 - x_{j1} - x_{j2} = x_{j3} + \cdots + x_{jn} \\
&\qquad\qquad \vdots \\
&x_{i(n-1)} + x_{in} \leq 1 - x_{j1} - \cdots - x_{j(n-1)} = x_{jn} \\
&x_{in} \leq 1 - x_{j1} - \cdots - x_{jn} = x_{j1} + \cdots + x_{jn} - 1
\end{aligned}$$

By adding the above inequalities, we obtain

$$\begin{aligned}
x_{i1} + 2x_{i2} + 3x_{i3} + \cdots + nx_{in} \leq \\
x_{j1} + 2x_{j2} + 3x_{j3} + \cdots + nx_{jn} - 1.
\end{aligned}$$

This is exactly the associated constraint in the set of constraints (5). As a result, the solution space of the LP relaxation of ASF′ is included in that of ASF. To show that the inclusion is strict, consider the following fractional values for the decision variables corresponding with a couple $(i,j) \in A$: $x_{i1} = x_{i5} = 0.5$ and $x_{j4} = 1$. These values can be seen to respect the weak but not the strong formulation. □

**Proof of Theorem 1:** Since $1||\sum w_j T_j$ is a relaxation of $1|\beta|\sum w_j T_j$, the optimal value of $I'$ and any valid lower bound for this optimal value is considered as a valid lower bound for $I$. □

**Proof of Theorem 2:** The following equality holds:

$$\begin{aligned}
\text{TWT}^*(I) &= \sum_{j \in \mathcal{T}} w_j (C_j(S^*) - d_j) \\
&= \sum_{j \in N} w_j (C_j(S^*) - d_j) \\
&\quad - \sum_{j \in \mathcal{E}} w_j (C_j(S^*) - d_j) = \sum_{j \in N} w_j C_j(S^*) \\
&\quad - \sum_{j \in N} w_j d_j + \sum_{j \in \mathcal{E}} w_j (d_j - C_j(S^*)).
\end{aligned}$$

Recall that $\bar{lb}_{I''}$ is a valid lower bound on the value $\sum_{j\in N} w_j C_j(S^*)$ and $lb^E$ is a valid lower bound on the value $\sum_{j\in\mathcal{E}} w_j(d_j - C_j(S^*))$. □

**Proof of Theorem 3:** We argue that

$$\mathrm{LB}_0 = L_0(\lambda_{\mathrm{PV}}) = \hat{L}_1(\lambda_{\mathrm{PV}}, \mu_0) \leq \hat{L}_1(\lambda_{\mathrm{TPA}}, \mu_{\mathrm{TPA}}) \leq L_1(\lambda_{\mathrm{TPA}}, \mu_{\mathrm{TPA}}) = \mathrm{LB}_1.$$

The first inequality follows from Theorem 3 in [47], where it is shown that TPA generates a series of monotonically increasing lower bounds. The second inequality corresponds with Theorem 1. □

**Proof of Theorem 4:** $\mathrm{LB}_1(\mathrm{NO})$ is obtained by solving $\mathrm{LR}_{\mathrm{P}_1}$ with $A' = \emptyset$ and $A'' = A$, so with the same multipliers the problem associated with $\mathrm{LB}_1(\mathrm{NO})$ is a relaxation of the problem associated with $\mathrm{LB}_1(\mathrm{VSP})$. The multipliers are updated with TPA in both cases, and will indeed be the same for a given $k_{\max}$, so the theorem holds. □

**Proof of Theorem 5:** We introduce $g_{B_i}(\lambda, \mu, S)$ as follows:

$$g_{B_i}(\lambda, \mu, S) = \sum_{j\in B_i} (w_j - \lambda_j)T_j + \sum_{j\in B_i} \lambda_j(C_j - d_j) + \sum_{j\in B_i}\sum_{k\in\mathcal{Q}_j} \mu_{jk}(C_j + p_k - C_k).$$

Let $S_1^*$ be an optimal solution to $\mathrm{LB}_1^*$ and $S_2^* = (S_{B_1}^*, \ldots, S_{B_\kappa}^*)$ an optimal solution to $\mathrm{LB}_2^*$. The following result is derived.

$$\mathrm{LB}_1^* = g(\lambda^*, \mu^*, S_1^*) \leq g(\lambda^*, \mu^*, S_2^*) = \sum_{i=1}^{\kappa} g_{B_i}(\lambda^*, \mu^*, S_{B_i}^*) \leq \sum_{i=1}^{\kappa} g_{B_i}(\lambda_{B_i}^*, \mu_{B_i}^*, S_{B_i}^*) = \mathrm{LB}_2^*.$$

□

**Proof of Theorem 6:** If $\min_{\pi\in\Pi_{\mathcal{A}}}\{C_{\max}(\sigma_B|\pi)\}$ is greater than $\max_{j\in\mathcal{A}}\{\bar{\delta}_j\}$ then at least one job in $\mathcal{A}$ cannot be scheduled before its deadline and the schedule $(\sigma_B, \sigma_E)$ is not feasible. □

**Proof of Theorem 7:** If $\tau_2' = 0$, then $\Gamma_{jk}(t, \tau_1, \tau_2) = \hat{\Gamma}_{jk}(t, \tau_1, \tau_2, \tau_2')$ and the theorem holds based on [22]. If $\tau_2' > 0$, all jobs in $\mathcal{B}$ are shifted to the left at least $\tau_2'$ units. Also, $\tau_2$ equals zero because no job is shifted to the right. For all jobs $i \in \mathcal{B}'$ we have $C_i(S_1) \geq C_i(S_1') \geq d_i$. Consequently, $\tau_2' \sum_{i\in\mathcal{B}'} w_i \geq 0$ is a lower bound for the gain of rescheduling jobs in $\mathcal{B}$. The value $w_j \max\{0, t+p_j-d_j\} - w_j \max\{0, \hat{r}_j+p_j-d_j\}$ equals the gain of rescheduling job $j$ and the value $w_k \max\{0, \hat{r}_k + p_k - d_k\} - w_k \max\{0, t + \tau_1 + p_k - d_k\}$ equals the gain of rescheduling job $k$. By adding lower bounds for rescheduling gains of all jobs in $U = \mathcal{B} \cup \{j, k\}$, a lower bound for the gain of interchanging jobs $j$ and $k$ is obtained. □

**Proof of Theorem 8:** We examine under which conditions the jobs belonging to the set $U = \mathcal{B} \cup \{j, k\}$ do not violate any of their deadlines and/or precedence constraints. Precedence constraints are not violated because jobs $j, k \in E_B$ are deliberately chosen such that $\mathcal{Q}_k \cap \mathcal{Q}_j = \mathcal{Q}_k$ and job $j$ does not violate its deadline simply because $C_j(S_1') \leq C_j(S_1) \leq \bar{\delta}_j$.

Condition 1 ensures that job $k$ does not violate its deadline. We argue that $t = st_j(S_1) \leq \bar{\delta}_j - p_j$. If $\bar{\delta}_j - p_j \leq \bar{\delta}_k - \tau_1 - p_k$ holds, then we infer $C_k(S_1') = t + \tau_1 + p_k \leq \bar{\delta}_k$. Also, if $\Psi \leq \hat{\delta}_k$, then all unscheduled jobs including $j$ and $k$ are completed at or before $\hat{\delta}_k$. Note that $\hat{\delta}_k$ is preferred over $\bar{\delta}_k$ because $\bar{\delta}_k \leq \hat{\delta}_k$, thus condition 1 is less violated, and the inequality $\Psi \leq \hat{\delta}_k$ also implies $C_k(S_1') \leq \bar{\delta}_k$.

Condition 2 verifies that no job in $\mathcal{B}$ violates its deadline. On the one hand, if $\tau_2 = 0$, then no job in $\mathcal{B}$ is shifted to the right, which means $C_i(S_1') \leq C_i(S_1) \leq \bar{\delta}_i$ for each job $i \in \mathcal{B}$. On the other hand, if $\tau_2 > 0$ and $\Psi \leq \min_{i\in\mathcal{B}}\{\hat{\delta}_i\}$, then for all jobs $i \in \mathcal{B}$ we conclude: $C_i(S_1') \leq \Psi \leq \min_{i\in\mathcal{B}}\{\hat{\delta}_i\} \leq \hat{\delta}_i$. Again, $\hat{\delta}_i$ is preferred over $\bar{\delta}_i$ because of the same reasoning as for the preference of $\hat{\delta}_k$ over $\bar{\delta}_k$. □

**Proof Lemma 4:** Let $f$ have a global minimum at value $t^*$. Depending on the values of the parameters $\alpha, \beta, a$ and $b$, the function $f$ behaves differently. We discuss four possible cases for the parameter combinations to prove this lemma (see also Figure 12). In the two first cases, we assume that $\alpha \geq \beta$. Case (a): in this case, $a \leq b$, and then $f$ is constant on interval $[u, a]$ and is increasing on interval $[a, v]$, as shown in Figure 12(a). Case (b): $a > b$, $f$ is constant on interval $[u, b]$, decreasing on interval $[b, a]$ and increasing on interval $[a, v]$, in line with Figure 12(b). The following results are valid for these two cases: 1- If $u \leq a \leq v$ then $t^* = a$. 2- If $a < u$, $t^* = u$ because $f$ is always increasing on interval $[u, v]$. 3- If $a > v$, $t^* = v$ because $f$ is always decreasing on interval $[u, v]$. We conclude that $t^* = \min\{\max\{a, u\}, v\}$ for the first two cases.

In the next two cases, we assume that $\alpha < \beta$. Case (c): $a < b$, $f$ is constant for $[u, b]$, increasing for $[b, a]$ and decreasing for $[a, v]$, as shown in Figure 12(c). In this case, $t^*$ equals either $u$ or $v$. On the one hand, if $\alpha(b - \max\{a, u\}) \geq (\beta - \alpha)(\max\{v, b\} - b) \Rightarrow \alpha(\bar{v} - \bar{u}) \geq \beta(\bar{v} - b)$ then $f(v) \geq f(u)$ is inferred and $t^* = u$ is concluded. On the other hand, if $\alpha(\bar{v} - \bar{u}) < \beta(\bar{v} - b)$ then

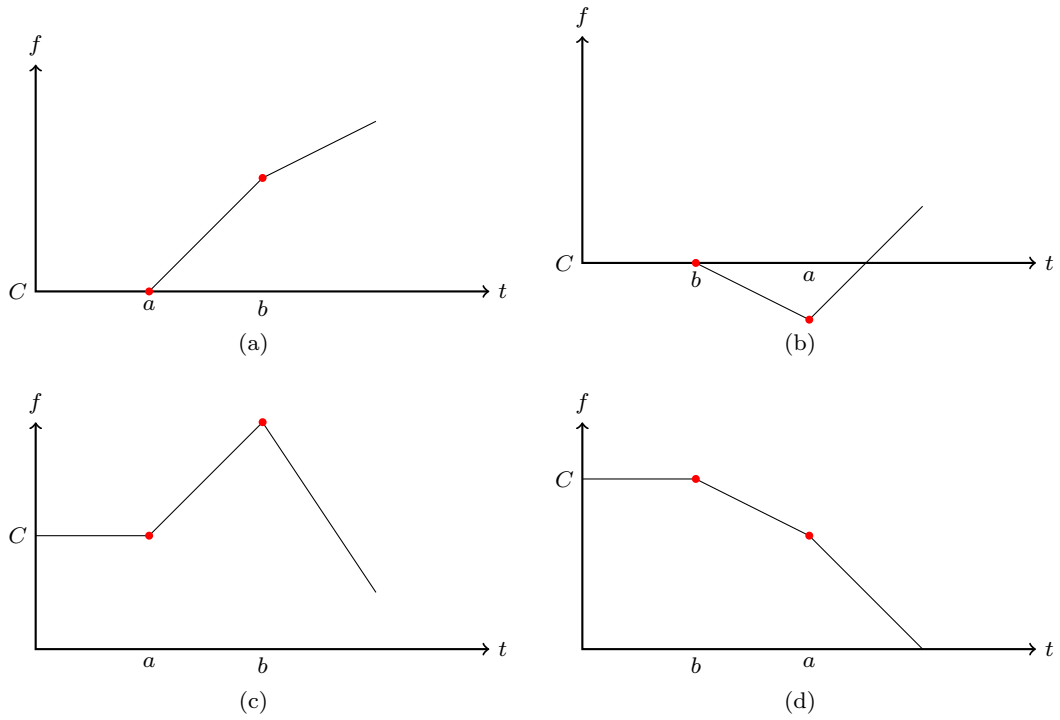


**Fig. 12:** Four possible cases for the parameter combinations in the proof of Lemma 4.

$t^* = v$ is concluded. Case (d): $a \geq b$, $f$ is constant for $[u, b]$ and decreasing for $[b, v]$; see Figure 12(d). We find that $t^*$ equals $v$ for this case. □

**Proof of Theorem 9 and Theorem 10:** Similar to the proof of Theorem 8. □

## References


1. T.S. Abdul-Razaq and C.N. Potts. Dynamic programming state-space relaxation for single-machine scheduling. *Journal of the Operational Research Society*, 39(2):141–152, 1988.
2. T.S. Abdul-Razaq, C.N. Potts, and L.N. Van Wassenhove. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*, 26:235–253, 1990.
3. M.S. Akturk and D. Ozdemir. An exact approach to minimizing total weighted tardiness with release dates. *IIE Transactions*, 32:1091–1101, 2000.
4. M.S. Akturk and D. Ozdemir. A new dominance rule to minimize total weighted tardiness with unequal release dates. *European Journal of Operational Research*, 135(2):394–412, 2001.
5. C. Artigues, S. Demassey, and E. Neron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. ISTE, 2007.
6. W. Bein, J. Kamburowski, and M. Stallmann. Optimal reduction of two-terminal directed acyclic graphs. *SIAM Journal on Computing*, 21(6):1112–1129, 1992.
7. H. Belouadah, M.E. Posner, and C.N. Potts. Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics*, 36(3):213–231, 1992.
8. G. Calinescu, C. Fernandes, H. Kaul, and A. Zelikovsky. Maximum series-parallel subgraph. *Algorithmica*, 63(1-2):137–157, 2012.
9. N. Christofides, R. Alvarez-Valdes, and J.M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29:262–273, 1987.
10. R.W. Conway, W.C. Maxwell, and L.W. Miller. *Theory of Scheduling*. Addison Wesley, Reading, MA, 1967.
11. D. Debels and M. Vanhoucke. A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55(3):457–469, 2007.
12. E. Demeulemeester, M. Vanhoucke, and W. Herroelen. Rangen: A random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6:13–34, 2003.
13. M.E. Dyer and L.A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26(23):255–270, 1990.
14. M.L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.
15. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
16. V. Gordon, E. Potapneva, and F. Werner. Single machine scheduling with deadlines, release and due dates. *Optimization*, 42(3):219–244, 1997.

17. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
18. A.M.A. Hariri and C.N. Potts. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics*, 5(1):99–109, 1983.
19. M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.
20. J.A. Hoogeveen and S.L. van de Velde. Stronger Lagrangian bounds by use of slack variables: Applications to machine scheduling problems. *Mathematical Programming*, 70:173–190, 1995.
21. T. Ibaraki and Y. Nakamura. A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76(1):72–82, 1994.
22. A. Jouglet, P. Baptiste, and J. Carlier. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, chapter 13, Branch and Bound Algorithms for Total Weighted Tardiness. CRC Press, Boca Raton, FL, USA, 2004.
23. A.B. Keha, K. Khowala, and J.W. Fowler. Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, 56:357–367, 2009.
24. J. Kinable, T. Wauters, and G. Vanden Berghe. The concrete delivery problem. *Computers & Operations Research*, 48:53–68, 2014.
25. E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546, 1973.
26. E.L. Lawler. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.
27. E.L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Algorithmic Aspects of Combinatorics*, 2:75–90, 1978.
28. J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. In *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977.
29. G.B. McMahon and C.J. Lim. The two-machine flow shop problem with arbitrary precedence relations. *European Journal of Operational Research*, 64(2):249–257, 1993.
30. G. Nemeth, I. Lovrek, and V. Sinkovic. Scheduling problems in parallel systems for telecommunications. *Computing*, 58:199–223, 1997.
31. R. Nessah and I. Kacem. Branch-and-bound method for minimizing the weighted completion time scheduling problem on a single machine with release dates. *Computers & Operations Research*, 39(3):471–478, 2012.
32. Y. Pan. An improved branch and bound algorithm for single machine scheduling with deadlines to minimize total weighted completion time. *Operations Research Letters*, 31(6):492–496, 2003.
33. Y. Pan and L. Shi. Dual constrained single machine sequencing to minimize total weighted completion time. *IEEE Transactions on Automation Science and Engineering*, 2(4):344–357, 2005.
34. M.L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2008.
35. M.E. Posner. Minimizing weighted completion times with deadlines. *Operations Research*, 33:562–574, 1985.
36. C.N. Potts. A lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time. *Management Science*, 31(10):1300–1311, 1985.
37. C.N. Potts and L.N. Van Wassenhove. An algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *European Journal of Operational Research*, 12(4):379–387, 1983.
38. C.N. Potts and L.N. Van Wassenhove. A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33(2):363–377, 1985.
39. B.A. Shirazi, K.M. Kavi, and A.R. Hurson, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, 1995.
40. D.R. Sule. *Production Planning and Industrial Scheduling: Examples, Case Studies and Applications*. CRC Press, 2007.
41. F. Talla Nobibon and R. Leus. Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers & Operations Research*, 38(1):367–378, 2011.
42. S. Tanaka and S. Fujikuma. A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. *Journal of Scheduling*, 15:347–361, 2012.
43. S. Tanaka, S. Fujikuma, and M. Araki. An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling*, 12:575–593, 2009.
44. S. Tanaka and S. Sato. An exact algorithm for the precedence-constrained single-machine scheduling problem. *European Journal of Operational Research*, 229(2):345–352, 2013.
45. L. Tang, H. Xuan, and J. Liu. Hybrid backward and forward dynamic programming based Lagrangian relaxation for single machine scheduling. *Computers & Operations Research*, 34(9):2625–2636, 2007.
46. J. Valdes, R. Tarjan, and E. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982.
47. S.L. van de Velde. Dual decomposition of a single-machine scheduling problem. *Mathematical Programming*, 69(1-3):413–428, 1995.
48. J. Van den Akker, G. Diepen, and J. Hoogeveen. Minimizing total weighted tardiness on a single machine with release dates and equal-length jobs. *Journal of Scheduling*, 13:561–576, 2010.
49. J. Xu and D. Parnas. Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Transaction on Software Engineering*, 16(3):360–369, 1990.